UNIVERSITY OF
LIVERPOOL

Efficient Combinatorial Algorithms for DNA Microarray Design

Thesis submitted in accordance
with the requirements of the University of Liverpool
for the degree of Doctor in Philosophy
by

Ying Li

Thesis Advisors
Dr. Prudence Wong
Prof. Leszek Gąsieniec

The work was done at
Department of Computer Science
University of Liverpool
January, 2008

Abstract

The advent of efficient genome sequencing tools and high-throughput experimental biotechnology has led to enormous progress in the life science. DNA microarray is among the most important innovations. It allows to measure the expression for thousands of genes simultaneously by analysing the hybridisation data. Such measurements have been proved to be invaluable in understanding the development of diseases such as cancer. However, the analysis of data is non-trivial since the hybridisation data relies on the quality of DNA microarray. High quality DNA microarray will lead to more efficient hybridisation and stronger signal and reliability. The reliability of data is essential. Thus, the development of novel algorithms and techniques for DNA microarray design is crucial.

This thesis considers a number of combinatorial issues in selecting, placing, and synthesising probes during the DNA microarray design process. A probe is a specific sequence of single-stranded DNA or RNA, typically labelled with a radioactive or fluorescent tag, which is designed to bind to, and thereby identify, a particular segment of DNA (or RNA). The probe selection problem we studied is to find for each gene sequence a unique probe such that every gene in the given dataset can be identified. However, due to homology, sometimes a gene does not have a unique probe, then we use a small number of non-unique probes to identify a gene. The challenge of the problem is that there are many candidate probes in a gene sequence and we have to find the right one (or a small subset) efficiently. A randomised probe selection algorithm for DNA microarray design is proposed. The algorithm overcomes some existing algorithms demanding optimal probes by

exhaustive search. We implement the randomised probe selection algorithm and develop a probe selection software RANDPS. Investigations using several real-life microarray datasets show that algorithm is able to find high quality probes.

Nevertheless, the number of the probes selected might be too large for placing in a single microarray, thus minimising the number of probes is an important objective, since it is proportional to the cost of the microarray experiment. Therefore, we investigate the string barcoding problem in which a set of non-unique probes is given and the probes have to be chosen from the given set of probes. The objective is to use an appropriate combination of probes with minimum cardinality such that all genes in the dataset can be distinguished. An almost optimal $O(n|\mathcal{S}|\log^3 n)$-time approximation algorithm for the considered problem is presented. The approximation procedure is a modification of the algorithm due to Berman *et al.* [10] which obtains the best possible approximation ratio $(1 + \ln n)$. The improved time complexity is a direct consequence of more careful management of processed sets, use of several specialised graph and string data structures, as well as tighter time complexity analysis based on an amortised argument.

After probes are selected, they are then synthesised on the microarrays by using a light-directed chemical process in which unintended illumination may contaminate the quality of the microarray experiments. Border length is a measure of the amount of unwanted illumination and the objective of this problem is to minimise the total border length during probe synthesis process. This problem is believed to be NP-hard and approximation of the BMP problem in asynchronous synthesis is studied. As far as we know, this is the first result with proved performance guarantee. The main result is an $O(\sqrt{n}\log^2 n)$-approximation, where n is the number of probes to be synthesised. In the case where the placement is given in advance, we show that the problem is $O(\log^2 n)$-approximable. A related problem called agreement maximisation problem (MAP) is also considered in this chapter. In contrast to BMP, we show that MAP admits a constant approximation even when placement is not given in advance.

Acknowledgement

To start with, I would like to thank my supervisors Dr. Prudence Wong and Prof. Leszek Gąsieniec for their advice throughout the duration of my PhD studies. Without their help this work would not be possible. They provided a lot of support, advice and research ideas. I was always welcomed to discuss my current work and future direction of research explorations.

I have also benefited from discussions with Derek Gardener, Christine Gosden and Dawn Jones at School of Cancer Studies in the University of Liverpool, Andy Cossins at Department of Biological Sciences in the University of Liverpool, Qin Xin at Simula Research Lab, Norway and Fencol Yung at Department of Computer Science in the University of Hong Kong. The meetings with them were enjoyable and very inspiring.

I am also grateful to Prof. Maxime Crochemore and Dr. Igor Potapov for agreeing to take part in the PhD examination process.

Last but not least, many thanks go to my parents Baoyou Li and Shuqing Lu for their ever-lasting love, encouragement and support.

Contents

List of Figures

List of Tables

Notation Glossary

Page	*Notation*	*Meaning*
12	D	A deposition sequence
13	M	A mask used in the DNA microarray synthesis process
33	g	A gene sequence
33	p	A probe
33	$\mathcal{G}$	A set of gene sequences $\{g_1, \ldots, g_{\lvert\mathcal{G}\rvert}\}$
33	$\mathcal{P}$	A set of probes $\{p_1, \ldots, p_{\lvert\mathcal{P}\rvert}\}$
33	ℓ	Length of a probe
34	T_m	Melting temperature
35	ΔG	Free energy
37	$g \in \mathcal{G}$	g is a member of $\mathcal{G}$
37	$\mathcal{G} - \{g\}$	Remove g from $\mathcal{G}$
37	$H(p, q)$	Hamming distance between probes p and q
53	$[i..j]$	The set of consecutive integers $\{i, i+1, \ldots, j-1, j\}$
53	$T \subseteq [0..1]$	T is a subset of $[0..n-1]$
53	$\overline{T}$	The complement ($[0..n-1] \setminus T$) of T
53	$\lvert T\rvert$	The cardinality of a set T
53	$P(T)$	The power set of a set $T \subseteq [0..n-1]$
54	$\mathcal{S}$	$\mathcal{S}$ is a subset of $P([0..n-1])$
54	$\mathcal{T}$	A collection of sets
54	$\overset{\mathcal{T}}{\equiv}$	Equivalence relation
54	E_x	An equivalence class
54	$E(\mathcal{T})$	The set of equivalence classes produced by the equivalence relation $\overset{\mathcal{T}}{\equiv}$
54	$H_{\mathcal{T}}$	Entropy function
55	$\otimes$	Combination of equivalence relations
55	$B(i, l)$	A set of consecutive integers $[(i-1) \times n/2^l..i \times n/2^l - 1]$
56	$\emptyset$	The empty set
57	$\mathcal{H}$	A hierarchical data structure

Page	*Notation*	*Meaning*
57	D_s	A binary tree structure representing a set S
57	$l(v)$	The left child of a node v in a tree
57	$r(v)$	The right child of a node v in a tree
59	$N(v)$	The name of a node v
61	SL	Structured list of equivalence classes
62	$\vec{G}$	A directed acyclic graph
69	$G = (V, E)$	The graph G with vertex set V and edge set E
69	ϕ	A placement of a set probes
69	ε	An embedding of a set of probes
70	ϕ^*	The optimal placement of a set probes
70	ε^*	The optimal embedding of a set of probes
69	$\text{conf}_\varepsilon(p, q)$	The conflict between the embeddings of probes p and q
69	$\text{share}_\varepsilon(p, q)$	The share between the embeddings of probes p and q
69	$\text{BL}(\phi, \varepsilon)$	The border length of a placement ϕ and an embedding ε
70	$\text{A}(\phi, \varepsilon)$	The agreement of a placement ϕ and an embedding ε
70	$LCS(a, b)$	The longest common subsequence of two sequences a and b
70	$SCS(a, b)$	The shortest common supersequence of two sequences a and b
70	$\text{dist}(a, b)$	The LCS distance of two sequences a and b
71	U	A set of k sequences $U = \{U_1, U_2, \ldots, U_k\}$
71	U'	An alignment of U
71	$\delta(a, b)$	A distance function of two sequences a and b
71	$w(a, b)$	A weight function of two sequences a and b
71	$\text{SP}(U', w)$	The the weighted sum-of-pair score of alignment U' and weight function w
72	$d(a, b)$	The edit distance between two sequences a and b
74	G_c	A weighted complete graph
81	$\tilde{Q}$	An approximate TSP
76	Q^*	The optimal TSP
82	τ	A tree
82	$parent(p)$	The parent of a probe p
82	$r(p)$	The right neighbour of a probe p
82	$b(p)$	The bottom neighbour of a probe p

Chapter 1

Introduction

Computational biology is concerned with the development of novel and efficient algorithms that can be proved to work on problems inspired by biology, such as multiple sequence alignment [11, 37, 74, 76, 81] and sequence homology search [2, 7, 22, 61, 72]. Computational biology is incredibly rich in terms of the number and variety of combinatorial problems it encompasses. This doctoral thesis has been motivated by several combinatorial problems arising from DNA microarray design.

This chapter is organised as follows. We first present some related biological background in Section 1.1. Then, we highlight the DNA microarray technology in Section 1.2. Further in Section 1.3, the problems studied in this thesis are introduced and are followed by the overview of the contribution of this thesis in Section 1.4.

1.1 Biological background

The section starts with the concepts of DNA, RNA, gene, gene expression, and hybridisation process.

1.1.1 DNA, RNA, gene, and gene expression

DNA (Deoxyribose Nucleic Acid) is a double-stranded molecule twisted into a helix like spiral staircase (see Figure 1.1). The DNA double helix is one of the greatest

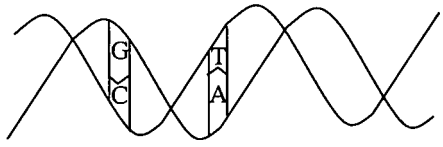

Figure 1.1: A DNA molecule. Nucleotide G is complementary to C, and T is complementary to A.

scientific discoveries which was first described by Watson and Crick [94]. Each strand of DNA, comprised of a sugar-phosphate backbone and attached nucleotides, is connected to a complementary strand by non-covalent hydrogen bonding between paired nucleotides. The four nucleotides are adenine (A), thymine (T), cytosine (C) and guanine (G). The nucleotides A and T are connected by two hydrogen bonds. The nucleotides G and C are connected by three hydrogen bonds. By Watson-Crick complementarity, nucleotide A is complementary to T and G is complementary to C. A ***gene*** is a specific region of DNA that represents a fundamental unit of inheritance.

Ribonucleic Acid (RNA) is a nucleic acid, consisting of a ribose, a phosphate backbone and four different nucleotides: adenine (A), cytosine (C),guanine (G) and uracil (U). The first three are the same as those found in DNA, but in RNA, thymine (T) is replaced by uracil (U) as the base complementary to adenine (A). RNA is usually single stranded, while DNA is normally double stranded. RNA is *transcribed* from DNA by enzymes called RNA polymerases. Once the information from the appropriate gene of DNA has been transcribed into RNA, the RNA serves as a messenger (mRNA). The information in mRNA is then used to direct the formation of a specific protein by *translation*. The process of producing a protein from the information stored in DNA is known as ***gene expression*** [88] .

1.1.2 Hybridisation process

Under normal conditions, a DNA molecule is composed of two strands. When a DNA molecule is heated, the hydrogen bonds disappear, and the two strands drift

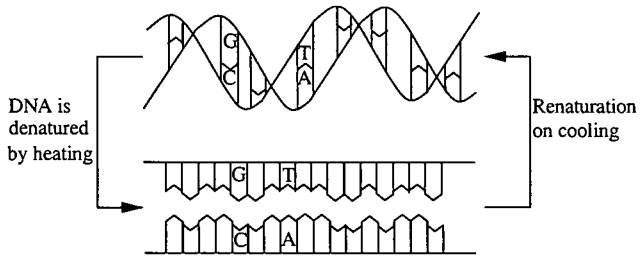

Figure 1.2: When a DNA molecule is heated, the two strands drift apart. When the temperature cools down, the double helix reappear by hybridisation.

apart (see Figure 1.2). When the temperature cools down, hydrogen bonds form between complementary bases in the strands. These bonds are formed in places where a match (or at least a partial match) exists. If these bonds begin to form in corresponding parts of two strands, they will quickly and completely bind together so that the double helix reappear. However, this is not guaranteed to happen. Bonds can form even between strands of different DNA molecules or strands of different length, which is known as *cross-hybridisation* .

1.2 DNA microarray technology

DNA microarray technology has become one of the most widely used tools in genomic study. It is used for performing a large number of hybridisation experiments simultaneously. It has been proved to benefit areas including gene discovery, disease diagnosis, and multi-virus discovery.

DNA microarray was proposed simultaneously and independently by Bains and Smith [8], Drmanac *et al.* [27] and Lysov *et al.* [63]. The inventors of DNA microarray suggested using it for DNA sequencing, and the original name for this technology was *DNA Sequencing by Hybridisation (SBH).* A DNA microarray[32] is a small plastic or glass slide which consists of an ordered series of spots containing short fragments of DNA which are called *probes.* A probe is a single-stranded

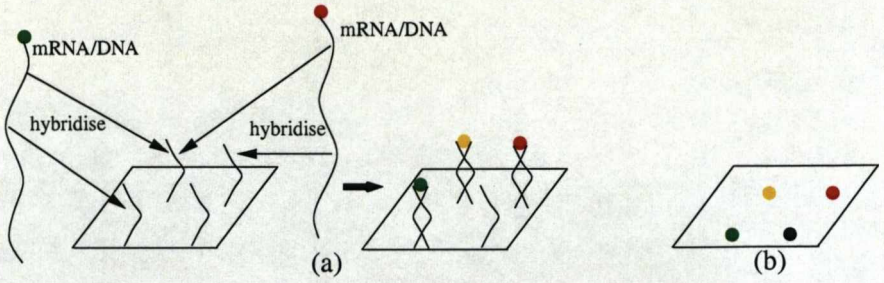

Figure 1.3: (a) A probe will hybridise to a target if there is a substring of the target that is the Watson-Crick complement of the probe. (b) Red, Green, Yellow: Probe hybridised to red-labelled, green-labelled, both red and green labelled mRNA/DNA sequence. Black: no hybridisation occurred.

fragment of a DNA or RNA, which acts as its fingerprint (a.k.a signature). Fingerprinting is the technique of identifying or confirming specific DNA fragments. Usually, a probe is $20-70$ nucleotides (nt) long. Through the use of highly accurate robotic spotters, over $30,000$ spots can be placed on one slide [86].

The most common application of DNA microarray is gene expression analysis. In this technique, mRNA/DNA sequences are labelled with two different fluorochromes (generally the green cyanine 3 and the red cyanine 5) before being hybridised to a microarray. The probes will hybridise to the mRNA/DNA sequence which contains an mRNA/DNA fragment that is complementary to the probe. For example, a probe consisting of ACCGTG will hybridise to a target DNA sequence CCC***TGGCAC***CTA since the probe is complementary to the substring TGGCAC of the target (see Figure 1.3 (a)). After hybridisation, a scanner will record the intensity of the fluorescence emission signals that is proportional to the degree of hybridisation that haveEf occurred. The spots containing probes that hybridisation has occurred are identifiable by different colours. Areas on the array with probes hybridised to red-labelled mRNA/DNA sequence will produce red spots. On the contrary, probes hybridised to green-labelled sequence will produce green spots. If probes hybridise to both red and green labelled sequence, the spot will have both

colour signals, as indicated with the yellow colour. When no hybridisation occurred, the spots will be visualised as black (see Figure 1.3 (b)). The colour can vary depending on the mixture of fluorescence molecules. The microarray data are then analysed using specific softwares that enables clustering of genes with similar expression patterns, assuming that they share common biological functions.

1.3 Studied problems and previous work

DNA microarray design raises a number of challenging combinatorial problems, such as deposition sequence design [53, 79, 93], manufacturing quality control [4, 44, 87], microarray gene expression data clustering [77, 99–101], probe selection [35, 46, 54, 59, 62, 78, 80, 83, 90, 96] and probe placement and synthesis [17–19, 39, 47–51]. In this thesis, we study three problems arising from DNA microarray design, namely, *probe selection*, *string barcoding* which is a variant of probe selection, and *border minimisation* arising from probe placement and synthesis.

Depending upon the application of microarray, the hybridisation experiments are conducted using either unique probes or a combination of non-unique probes (probes hybridise to more than one target). In the case of unique probes, we can infer the presence of a target in a sample if the target sequence hybridises to its corresponding unique probe on the microarray. Therefore, unique probes should be selected which leads to the probe selection problem. In the case of non-unique probes, we will observe the hybridisation of all probes incident to the target present in the sample. If the probe set is chosen carefully, the hybridisation pattern is distinct from all other patterns. This gives rise to the string barcoding problem.

The probe selection problem we studied is to find for each gene sequence a unique probe such that every gene in the given dataset can be identified. However, due to homology[1], sometimes a gene does not have a unique probe, then we use a

[1]Homology is the similarity in DNA or protein sequences between individuals of the same species or among different species.

small number of non-unique probes to identify a gene. The challenge of the problem is that there are many candidate probes in a gene sequence and we have to find the right one (or a small subset) efficiently.

Nevertheless, the number of the probes selected might be too large for placing in a single microarray, thus minimising the number of probes is an important objective, since it is proportional to the cost of the microarray experiment. Therefore, we investigate the string barcoding problem in which a set of non-unique probes is given and the probes have to be chosen from the given set of probes. The objective is to use an appropriate combination of probes with minimum cardinality such that all genes in the dataset can be distinguished.

After probes are selected, the next stage is to place and synthesise the probes on the microarray where border minimisation problem (BMP) is brought to our attention. This is an optimisation problem to minimise the contamination during probe synthesis process. The following sections summarise the studied problems and some previous work.

1.3.1 Probe selection

DNA microarray is an efficient tool for making a qualitative statement about the presence or absence of biological target sequences in a sample. For example, we have a database of the DNA sequences for a known family of viruses and the problem is to identify an unspecified virus whose DNA sequence is present in the database. What we need is a set of hybridisation tests based on good selection of probes such that on every known family, the set of answers (red, green, yellow or black signal on the microarray) that we receive is unique with respect to any other virus in the database. Therefore, the probe should bind only to its corresponding sequence, and not to any other sequence available in the database. If this is the case, we say that the probe is *unique*.

The quality of the probe selection process can be expressed by the proportion

of unique probes found by an algorithm so that a genomic sequence can be distinguished from others in the database. In such an experiment the presence or absence of targets is determined by observing whether selected probes bind to their corresponding targets. The problem is to select a probe set that is able to uniquely identify targets while containing a small number of probes. However, in cases where the existence of a unique probe is unlikely, e.g., in the context of a large family of closely homologous genes, the use of a limited number of non-unique probes is still desirable.

The probe selection problem considered in this thesis is to find a small number of good probes with specified length for every gene in the genome, that satisfies

1. *Homogeneity* - melting temperature for every probe should be within some pre-defined range, to make sure that the probes are able to hybridise to their intended targets at about the same experimental temperature.

2. *Sensitivity* - probes prone to self-complementarity should be eliminated, to make sure that the probes hybridise to their intended targets rather than itself.

3. *Specificity* - probes should be unique to each gene in the genome on the basis of the Hamming Distance [38] as the similarity measure[2]. Hamming distance becomes a powerful tool for determining closeness/similarity and recently has been adopted as the specificity measure [59, 78, 90].

The specificity check is computationally expensive and takes the most time in the probe selection process. The brute force approach for specificity checking scans through the whole length-n genome for every length-ℓ probe and determines if the Hamming distances are large enough. Such a process is expensive and requires $O(\ell n^2)$ time. For example, brute force specificity checking would take

[2]For two strings s and t of the same length, the Hamming distance $H(s,t)$ is the number of positions where the characters at corresponding positions of the two strings differ. For example, if s = 00010101, t = 00011010, then $H(s,t) = 4$.

about 72 hours for S.pombe genome of length 7.1×10^6 bps and is thus impractical for large genomes. A good probe selection algorithm should be both time and space efficient.

To further improve the quality of the probe selected, we use additional constraints, including the rules described by [62] and those used in the Affymetrix probe selection criteria: (1) no single base (A, T, C or G) exceeds 50% of the probe size; (2) the length of any contiguous sequence of As and Ts or Cs and Gs region is less than 25% of the probe size; (3) GC-content is between 40% and 60% of the probe sequence (GC-content is the percentage of nucleotides which are G or C in the sequence). We refer to these constraints as *Quantitative criteria* .

Due to its significance, probe selection attracts a lot of attention [35, 46, 54, 59, 62, 78, 83, 90, 96]. Various probe selection algorithms have been developed in recent years. Good probe selection algorithms should produce a small number of candidate probes. Efficiency is also crucial because the data involved is usually huge. Most existing algorithms usually select probes by filtering, which is usually not selective enough and quite a large number of probes are returned.

Previous work - selection criteria. Lockhart *et al.* [62] were among the first to study the probe selection problem. The quantitative criteria they proposed are widely used [14, 59, 82, 83, 90, 92], with some minor variations.

Homogeneity and specificity were also used in their algorithm, though the exact algorithm has not been published. Homogeneity is used in almost all existing algorithms, in which is usually measured by the nearest neighbour model (NNM). Kaderali and Schliep [46] focus on melting temperature (T_m) and compute the optimal (the best) probe using suffix trees and dynamic programming. However, this is too slow, especially for large genomes, e.g., it takes 2 weeks to design a probe set for the whole yeast genome. A different formula was also used in [14, 96] to calculate T_m. Other work like [68] also only focuses on criteria related to thermodynamic evaluation. It is generally agreed that T_m and free energy can be used as parameters to evaluate probe hybridisation behaviour and have been shown to be

useful [59]. Some researchers [73, 98] argue that thermodynamic criteria may not be adequate for microarray analysis, we leave this decision to biologists while we mainly provide a computational tool to design probes using thermodynamic criteria.

As for specificity, there are two major measurements: Hamming distance [59, 78, 90] and BLAST search [14, 82, 83, 92, 96]. Using BLAST [5] (http://www.ncbi.nih.gov/blast/), the algorithms assume the search is done in advance and the results passed as input. Thus, the computation time depends on the number of sequences in the BLAST database; e.g., the algorithm by [83] takes from 4 to 12 hours to design up to three 45nt probes per gene for most of the bacterial genome.

Sensitivity is also a popular consideration to avoid self-binding of probes selected. This may be done by checking the stability of the secondary structure formed (stable means not a good candidate). MFOLD [102], Vienna RNAfold [43] and Smith-Waterman [89] algorithms have been used in [14, 68, 83, 96] for this purpose. It is worth mentioning that recently there have been other softwares developed for predicting secondary structure, e.g., Sfold [26], UNAFOLD [66], though they are not yet employed directly in the context of probe selection. Other algorithms [59, 78, 82, 90, 92] directly check sensitivity by eliminating probes that are self-complementary.

Previous work - existing software. Based on the above three criteria, a number of algorithms have been proposed. Li and Stormo [59] used a fast approximate matching search algorithm Myersgrep [71] for uniqueness checking. However, the algorithm is still not fast enough for computing probes of large genome sets. It takes almost four days to design a length-24 probe set for *Saccharomyces cerevisiae* genome (12M nt with about 6000 genes). Rahmann [78] presented a fast algorithm eliminating candidates that have a long common factor with other genes. This algorithm allows selection of probes for large genomes like *N.crassa* with total size 43MB in 4 hours on a Compaq ES40 (833 MHz) with 16GB memory. However, the approach only designs short probes and requires a lot of space during compu-

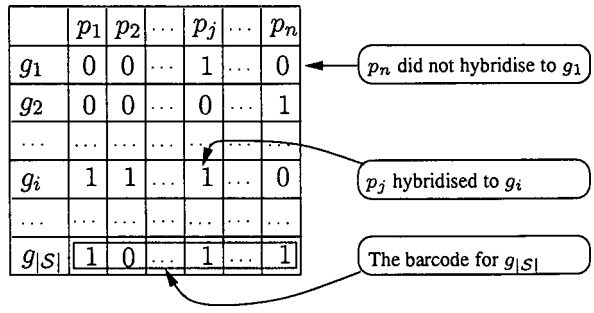

Figure 1.4: A barcode is a vector consisting of the hybridisation values between a gene and each probe. 1 : Hybridisation, 0 : No hybridisation.

tation. Sung and Lee [90] attempted to reduce the time complexity by using several filtering steps and exploiting the Pigeon Hole Principle [15] to avoid redundant comparisons. A length 50nt probe set for *N.crassa* can be generated in 3.5 hours on SunFire Workstations (700MHz) with 4GB memory.

Relogio *et al.* [82] proposed a modified version of the Gene Skipper software; the specificity check only considers perfect matches ignoring possible mismatches which may still result in probes that are non-specific and bind to other sequences in addition to the target. Tolonen *et al.* [92] also only considered perfect matches; specificity checking requires no region of self complementarity of five or more bases at either end. Wright and Church [96] proposed an algorithm which terminates once good probes (not necessary optimal) are found. They also introduced an interesting concept to define probe sequence complexity based on the Lempel-Ziv (LZ) compression algorithm [57]. Independently, this idea was also employed by [14].

1.3.2 String barcoding

The string barcoding problem , discussed by Rash and Gusfield [80], is used for the identification of genomic sequences (targets), such as viruses or bacteria, from among a set of known targets. Applications of this technique range from efficient pathogen identification in medical diagnosis to monitoring of microbial communi-

ties in environmental studies [12]. The wide range of applications lead to the same methodological problem which is to determine the presence or absence of one target in a biological sample.

Targets identification is performed by synthesising the Watson-Crick complements of the probes on a microarray [32], then hybridising to the array the fluorescently labelled DNA extracted from the unknown target. Under the assumption of perfect hybridisation stringency, the hybridisation pattern can be viewed as a string of $0's$ and $1's$ where 1 represents a probe that hybridises to a target. This 0 and 1 patten is referred to as the *barcode* of the target (see Figure 1.4). For unambiguous identification, probes must be selected such that each genomic sequence has a distinct barcode.

In this thesis, we study a variant of the string barcoding problem in which the probes have to be chosen from a given set of probes of cardinality n, such that an appropriate combination of probes with minimum cardinality is used to distinguish between all gene sequences in a dataset S. The string barcoding problem can be also obtained from more general *test set problem*, see, e.g., [10] by fixing appropriate parameters.

Previous work. Borneman *et al.* [12] were among the first to study the string barcoding problem. They proposed two efficient algorithms based on simulated annealing and Lagrangian relaxation for selecting a minimal probe set to be used in the oligonucleotide fingerprinting of rDNA clones by hybridisation experiments on DNA microarrays. However, the running time of these algorithms does not scale well with the number and the length of the genomic sequences mainly because of its requirement of large memory space. The string barcoding problem has been popularised by Rash and Gusfield [80]. In their paper, they proposed an *integer programming* approach to express the minimisation problem and represented strings by using suffix trees. They also stated that the constrained version of string barcoding, where the maximum length of each probe is bounded by a constant with the

alphabet size at least 3, is NP-hard. They also stated the approximability of string barcoding as an open problem. In [56], Lancia and Rizzi showed that the string barcoding problem is as hard to approximate as the *set cover* problem. Furthermore, they showed that the constrained version of string barcoding with probes of bounded length is also difficult to approximate. Finally they proved that both constrained and unconstrained string barcoding are NP-complete even for binary alphabets.

Klau *et al.* [54] presented an approach to select a minimal probe set for the case of non-unique probes in the presence of a small number of multiple targets in the sample. Their approach is based on integer programming together with a branch-and-cut algorithm. Their preliminary implementation is capable of separating all pairs of targets optimally in a reasonable time and achieves a considerable reduction on the numbers of probes needed compared to previous greedy algorithms. DasGupta *et al.* [24, 25] proposed a greedy algorithm for robust string barcoding. Their method enabled probe selection based on whole gene sequences of hundreds of microorganisms of upto bacterial size on well-equipped work station. Berman *et al.* [10] proposed an $O(n^2|\mathcal{S}|)$ time approximation algorithm for test set problem (TS) with approximation ratio $(1 + \ln n)$. The approximability result in [10] holds for general test set problems which includes string barcoding as a special case. The algorithm proposed in [10] is a greedy algorithm in which the choice of the test set to be added at each step is determined by a suitable *entropy* function.

1.3.3 Border minimisation

Probes are synthesised on the microarray through the process called *very large-scale immobilised polymer synthesis* (VLSIPS) [29] . In each step, light is selectively allowed through a *mask* to expose *spots* in the microarray in order to activate the nucleotides in the spots. The patterns of the masks used and the sequence of the deposition nucleotides in the illumination define the ultimate sequence of nucleotides of the array spot. A mask consists of masked (blocking light) and unmasked (allow-

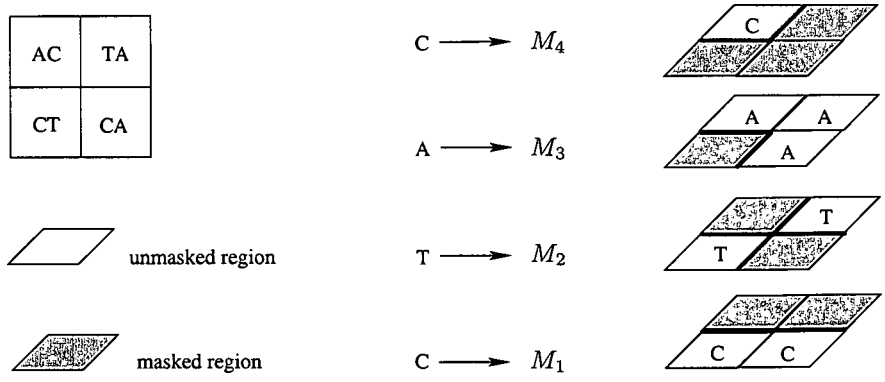

Figure 1.5: Synthesis of a 2×2 microarray. The deposition sequence $D = \text{CTAC}$ corresponds to four masks M_1, M_2, M_3, and M_4. Masked regions are shaded. The borders between masked and unmasked regions are represented by bold lines.

ing light) regions and induces deposition of a particular nucleotide (A, C, G or T) at its exposed array *spots*. The *deposition sequence* D corresponding to the sequence of masks is a supersequence of all probes in the array (see example in Figure 1.5).

DNA microarray synthesis consists of two phases, which are *probe placement* and *probe embedding*. Given a set of probes to be synthesised, probe placement is to place each probe to a unique spot in the microarray and probe embedding is the sequence of masked and unmasked steps used in the synthesis of each probe. For example, in Figure 1.6, the deposition sequence is $(ACGT)^3$ and the sequence (a) $A(-)^4C(-)^5T$ is a possible embedding of the probe ACT, where " $-$ " represents a space, and similarly for (b) to (d).

We distinguish two types of synthesis, namely, *synchronous* and *asynchronous* synthesis. In synchronous synthesis, each deposition nucleotide can only be deposited to the i-th position of the probes for a particular i, corresponding to a unique embedding of the probe (see Figure 1.6 (a)). In the case of asynchronous synthesis, there is no such restriction, allowing arbitrary embeddings (see Figure 1.6 (b) - (d)). For example, Figure 1.5 shows an asynchronous synthesis in which M_2 deposits a nucleotide to the second position of the sequence CT and the first position of TA.

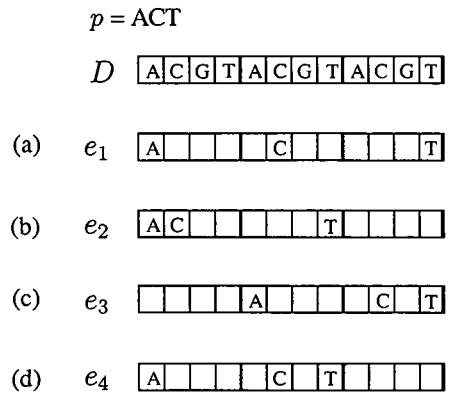

Figure 1.6: Embeddings of probe $p = ACT$ into deposition sequence $D = (ACGT)^3$. (a) Synchronous embedding of p into D. (b) - (d) Three different asynchronous embeddings of p into D.

Due to diffraction, internal reflection and scattering, spots on the *border* between masked and unmasked regions are often subject to unintended illumination [29]. This uncertainty produces unpredicted probes that can compromise experimental results. As microarray chip is expensive to synthesise, so it is usual that as many probes as possible are placed in a chip (i.e., as many entries are used), while unintended illumination has to be minimised. The magnitude of unintended illumination can be measured by the *border length* of the masks used. The border length of a mask is the number of borders shared between masked and unmasked regions. For example, in Figure 1.5, the border length of masks M_1, M_3, M_4 is 2 and M_2 is 4.

To reduce the amount of unintended illumination, one can exploit freedom in placing probes in the microarray during probe placement and choosing different probe embeddings. The *Border Minimisation Problem (BMP)* [39] is to find a placement of the probes on the microarray together with their embeddings in such a way that the sum of border lengths over all masks is minimised. It has been stated in [17, 18] that the problem is believed to be NP-hard because of the exponential number of possible placements, although we are not aware of an NP-hardness proof. For this reason, we focus on approximation algorithms for BMP in this thesis.

Previous work. The BMP problem has attracted a lot of attention [17–19, 39, 47–51] and most of the existing work is experimental in nature. As far as we are concerned, there is no known polynomial-time approximation algorithm for BMP with non-trivial performance guarantee.

BMP was first formally defined by Hannenhalli *et al.* [39]. They focused on synchronous synthesis and the only concern became probe placement. Their algorithm computes an approximated travelling salesman path (TSP) in the complete graph with nodes representing probes and edge costs representing the Hamming distance between the probes. The motivation for using a TSP tour is that consecutive probes in the tour are likely to be similar. The TSP tour is then placed on the microarray in a certain way called *threading*. Experiments show that threading is effective in reducing border length. Since then, other algorithms [18, 47–49, 51] have been proposed to improve the experimental results, most of which are based on ordering and partitioning.

Asynchronous probe embedding was introduced by Kahng *et al.* [49]. They studied a special case that the deposition sequence D is given and the embeddings of all probes are known except one. A polynomial-time dynamic programming algorithm was proposed to compute the optimal embedding of this single probe whose neighbours are already embedded. This algorithm is used as the basis for several approaches [17–19, 47–51] that have been shown experimentally to reduce the unintended illumination in terms of border length.

Apart from experimental results, there are few theoretical results. In [49], lower bounds on the total border length for synchronous and asynchronous BMP problem were given, based on Hamming distance, and Longest Common Subsequence (LCS), respectively. The asynchronous dynamic programming mentioned above computes the optimal embedding of a single probe in time $O(\ell|D|)$, where ℓ is the length of a probe and D is the deposition sequence. The algorithm can be extended to an exponential time algorithm to find the optimal embedding of all n probes, the corresponding time complexity is $O(2^n \ell^n |D|)$.

1.4 Contribution of the thesis

The studied problems in Section 1.3 are biologically closely related but require different techniques to solve. For the probe selection problem, randomisation is exploited, while for the string barcoding problem, we make use of several specialised graph and string data structures as well as amortised analysis. As for the BMP problem, the $O(\sqrt{n}\log^2 n)$-approximation result is based on a polynomial time reduction of the BMP problem to the weighted multiple sequence alignment problem (WMSA). An existing approximation of the minimum routing cost spanning tree problem which can be used to approximate WMSA is employed. The contribution of the thesis is summarised as follows.

In Chapter 3, we propose a new direction to tackle the probe selection problem and give an efficient algorithm based on randomisation to select a small set of probes and demonstrate that such a set of probes is sufficient to distinguish each sequence from all the other sequences. We implement the randomised probe selection algorithm and develop a probe selection software RANDPS. The software is available on the following website (http://www.csc.liv.ac.uk/~cindy/RandPS/RandPS.htm). The probe selection algorithm is tested via experiments on different genomes (E.coli, S.cerevisiae etc.) and the algorithm is able to output unique probes for most of the genes efficiently. In terms of accuracy of probe selection, RANDPS is able to find unique probes for up to 99% of genes in the whole genome. The other genes can be identified by a combination of at most two probes. This is a joint work with Leszek Gąsieniec, Paul Sant and Prudence W. H. Wong. A paper with the title "Efficient Probe Selection in Microarray Design" has been published in the *Proceedings of the 2006 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (IEEE CIBCB 2006)*, pp. 247–254 [34]. The full version is published in *Journal of Theoretical Biology*, 248(3), pp. 512–521, 2007 [35].

In Chapter 4, we study a variant of the string barcoding problem in which the probes have to be chosen from a given set of probes of cardinality n and the objec-

tive is to find a minimum set of probes that distinguish between all gene sequences in a given set of sequences $\mathcal{S}$. An almost optimal $O(n|\mathcal{S}|\log^3 n)$-time (in view of the fact that the size of the input to the studied problem is of size $\Omega(n|\mathcal{S}|)$) approximation algorithm for the considered problem is presented. The approximation procedure is a modification of the algorithm due to Berman *et al.* [10] which obtains the best possible approximation ratio $(1+\ln n)$, provided $NP \not\subseteq DTIME(n^{\log\log n})$. The improved time complexity is a direct consequence of more careful management of processed sets, use of several specialised graph and string data structures as well as tighter time complexity analysis based on an amortised argument. This is a joint work with Leszek Gąsieniec and Meng Zhang. A paper with the title "Faster Algorithm for the Set Variant of the String Barcoding Problem" [33] is expected to be published soon.

In Chapter 5, approximation of the BMP problem in asynchronous synthesis is studied. As far as we know, this is the first result with proved performance guarantee. The main result is an $O(\sqrt{n}\log^2 n)$-approximation, where n is the number of probes to be synthesised. In the case where the placement is given in advance, we show that the problem is $O(\log^2 n)$-approximable. A related problem called agreement maximisation problem (MAP) is also considered in this chapter. In contrast to BMP, we show that MAP admits a constant approximation even when placement is not given in advance. This is a joint work with Prudence W. H. Wong, Qin Xin and Fencol C. C. Yung. A paper with the title "Approximating Border Length for DNA Microarray Synthesis" [58] is expected to be published soon.

Chapter 2

Preliminaries

In this chapter, we present first some biological concepts related to DNA microarray design. Later, we provide basic definitions and notation that are used in the thesis.

2.1 Related biological concepts

This section starts with the concepts of free energy and melting temperature.

2.1.1 Free energy and melting temperature

Free energy is the amount of thermodynamic energy which can be viewed as a function of DNA hybridisation stability [75]. ***Melting temperature*** [84] of a probe is the temperature at which 50% of the probes and its perfect complement are in duplex. It can be considered as a parameter to evaluate probe hybridisation behaviour.

Since different mismatches (including insertion, deletion and mismatch) have different free energy and mismatch locations have different effects on the stability of DNA hybridisation, simple counting of mismatches could not determine the stability of DNA hybridisation structure. Free energy and melting temperature between a probe and a target on a DNA microarray could not be directly calculated, because DNA hybridisation behaviour on a microarray is not the same as that in solution and the parameters on a microarray are not available currently. Therefore, using

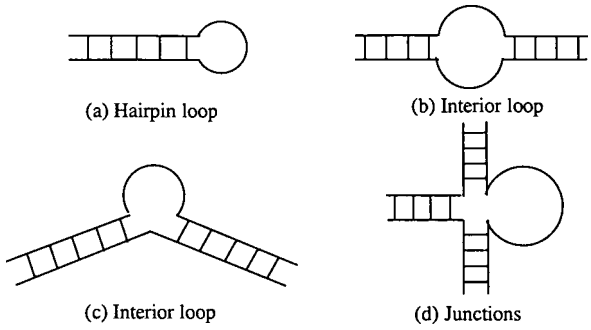

Figure 2.1: Types of loop and junction arrangements in RNA secondary structures.

thermodynamic parameters measured in solution could predict stability of probe hybridisation on microarrays approximately.

DNA oligonucleotide nearest-neighbour thermodynamic parameters are available [85] and they allow prediction of oligonucleotide DNA hybridisation energies. The calculation is approximate, but still useful. The details about the thermodynamic parameters used in this thesis can be found in Section 3.2.2.

2.1.2 DNA secondary structure

DNA secondary structure is the double-stranded regions of the molecule formed by folding the single-stranded molecule back on itself to form loops in the DNA structure. To produce these double-strand regions, a number of nucleotides downstream in the DNA molecule must be complementary to a number of nucleotides upstream so that Watson-Crick base paring between the complementary nucleotides G/C and A/T can occur. DNA secondary structure predictions are composed of base-paired and non-base-paired regions forming various types of loop and junction arrangements, as shown in Figure 2.1. The stem and the hairpin loop in Figure 2.1 (a) must generally be at least four nucleotides long. Interior loops form when the nucleotides in double-stranded region cannot form base pairs, as displayed in Figure 2.1 (b)-(c). Junctions, as shown in Figure 2.1 (d), may include two or more double-stranded

regions.

All types of DNA secondary structure analysis begin by the identification of *self-complementary* sequence regions. For single-stranded DNA molecule, it can potentially hybridise to itself to form DNA double strands [70]. During DNA microarray hybridisation experiments, the probe on the microarray should form a double helix with the complementary segment of the target DNA (from the sample investigated) rather than hybridising to itself, and thereby triggers a signal. Therefore, probes prone to self-complementarity should be avoided for DNA microarray design. The self-complementary criterion adopted in this thesis is explained in detail in Section 3.2.2.

2.1.3 Very large-scale immobilised polymer synthesis (VLSIPS)

Very large-scale immobilised polymer synthesis (VLSIPS) is a method that uses light to direct the simultaneous synthesis of many different chemical compounds. Synthesis occurs on a solid support. The pattern of exposure to light through a mask determines which regions of the support are activated for chemical coupling [29].

Activation by light results from the removal of photolabile protecting groups from selected area. After deprotection, the first of a set of nucleotides (each containing a photolabile protecting group) is exposed to the entire surface, but reaction occurs only with regions that were addressed by light in the preceding step. The wafer surface is then illuminated through a second mask, which activates a different region for reaction with a second nucleotide with a photolabile protecting group. Additional cycles of photodeprotection and coupling are carried out to obtain the desired set of products. The pattern of masks used in these illuminations and the sequence of reactions define the ultimate products and their locations.

Figure 2.2 illustrates the VLSIPS process to manufacture DNA microarrays. A microarray bears nucleotides that are blocked with a photolabile protecting group. Illumination of specific regions through a mask leads to photo-deprotection. Nu-

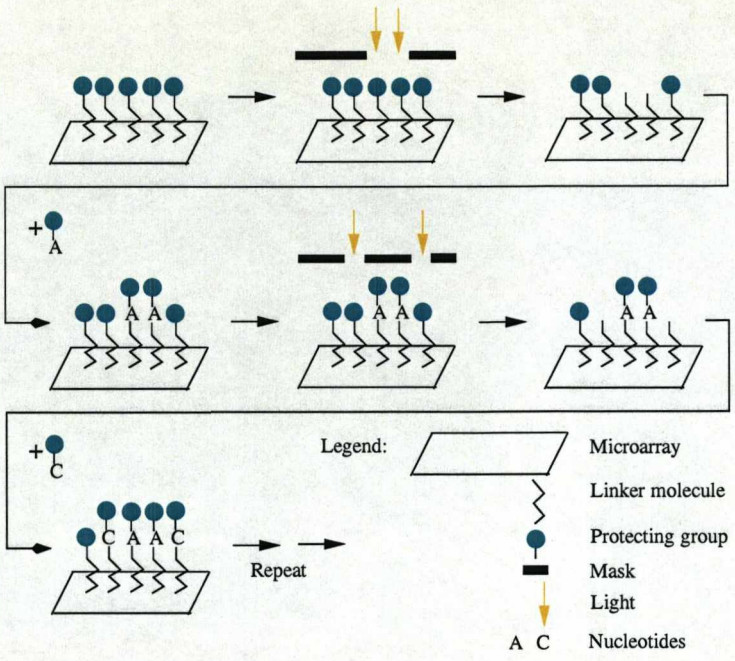

Figure 2.2: The VLSIPS process to manufacture the microarrays.

cleotides in the exposed regions of the microarray are now accessible for coupling. The first nucleotide A containing a photolabile protecting group is then attached. A different mask is used to photoactivate a different region of the microarray. A second nucleotide C with a photolabile protecting group is then added. The process of selective light-directed deprotection and nucleotide addition is repeated until all probes are synthesised on the microarray.

2.2 Basic algorithmic notation and definitions

In this section, several fundamental computational definitions are provided.

2.2.1 Algorithm, complexity theory, and asymptotic notation

An informal definition of an ***algorithm*** [23] would state that as a well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output. An algorithm is thus a sequence of computational steps that transform the input into the output. Harel [40] defines it as "an abstract recipe, prescribing a process which may be carried out by a human, a computer or by other means". An algorithm can be translated into instructions that are executable. Translating an algorithm into an imperative language involves building the solution using data structures such as variables and arrays and joining them with control structures such as conditional and loop statements. The main advantage of using imperative languages is that they have been designed with the computer architecture in mind, so the way the algorithm is executed by the computer is fairly straightforward.

The ***complexity theory*** is a branch of the theory of computation in computer science. The field is concerned with the scalability of algorithms, and the inherent difficulty in proving scalable algorithms for specific computational problems. There are two main topics under the computational complexity of an algorithm. *Time complexity* of an algorithm is the number of steps taken to solve an instance of the problem as a function of the size of the input, using the most efficient algorithm. *Space complexity* of an algorithm is a closely related concept, that measures the amount of space, or memory required by the algorithm.

Asymptotic notation is a way to express an algorithm's efficiency. Asymptotic notation is a mathematical notation used to describe the asymptotic behaviour of functions. When we have an ***asymptotic upper bound***, **O-notation** is used. For a given function $g(n)$, we denote by $O(g(n))$ the set of functions

$$O(g(n)) = \{f(n) : \text{there exists positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}.$$

Just as O-notation provides an asymptotic upper bound on a function, **Ω-notation** provides an ***asymptotic lower bound***. For a given function $g(n)$, we denote by $\Omega(g(n))$ the set of functions

$$\Omega(g(n)) = \{f(n) : \text{there exists positive constants } c \text{ and } n_0 \text{ such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0\}.$$

2.2.2 Probability and randomised algorithms

Probability is part of the conceptual core of modern computer science. Probabilistic analysis of algorithms, randomised algorithms and probabilistic combinatorial constructions have become fundamental tools for computer science. Probability provides a quantitative description of the likely occurrence of a particular event. In common usage, the word probability is used to mean the chance that a particular event (or set of events) will occur expressed on a linear scale from 0 to 1, also expressed as a percentage between 0 and 100%. A rare event has a probability close to 0 and a very common event has a probability close to 1.

Random is used to express uncertainty or lack of predictability. A random process is a repeating process whose outcomes follow no describable deterministic pattern, but follow a probability distribution. A ***randomised algorithm*** is an algorithm that employs some random or pseudorandom choices as part of its logic. In common practice, this means that the machine implementing the algorithm has access to a pseudorandom number generator. The algorithm typically uses the random bits as an auxiliary input to guide its behaviour, in the hope of achieving good performance in the "average case". Formally, the algorithm's performance will be a random

variable determined by the random bits, with hopefully good expected value. The "worst case" is typically so unlikely to occur that it can be ignored. For many applications, a randomised algorithm is either the simplest or the fastest algorithm available, and sometimes both.

2.2.3 Amortised analysis

An ***amortised analysis*** is a strategy for analysing a sequence of operations to show that the average cost per operation is small, even though a single operation within the sequence might be expensive [69]. The basic idea is that a worst case operation can not occur again for a long time, thus amortising its cost. The requirement is that the sum of the amortised costs of all operations in the sequence is greater than or equal to the sum of the actual costs. That is,

$$\sum_{1 \le i \le n} amortised(i) \ge \sum_{1 \le i \le n} actual(i) .$$

where $amortised(i)$ and $actual(i)$, respectively, denote the amortised and actual costs of the i-th operation in a sequence of n operations. Because of this requirement, we can use the sum of the amortised cost as an upper bound on the cost of any sequence of operations.

There are three most common techniques used in amortised analysis. *Aggregate analysis* determines the upper bound $T(n)$ on the total cost of a sequence of n operations. The average cost per operation is then $T(n)/n$, which is taken as the amortised cost of each operation, so that all operations have the same amortised cost. *Accounting method* determines the individual cost of each operation. When there are more than one type of operations, each type of operation may have a different amortised cost. The accounting method overcharges some operations early in the sequence, storing the overcharges as "prepaid credit" on specific objects in the data structure. The credit is used later in the sequence of operations to pay for operations that are charged less than they actually cost. *Potential method* is like

the accounting method in which we determine the amortised cost of each operation and may overcharge operations early to compensate for undercharges later. The potential method maintains the credit as the "potential energy" of the data structure as a whole instead of associating the credit with individual objects within the data structure.

2.2.4 Approximation algorithms

Approximation algorithms have been developed in response to the impossibility of solving a great variety of important optimisation problems [42]. Many problems of practical significance are NP-hard, thus we are unlikely to find polynomial-time algorithms for solving them optimally. However, it may still be possible to find near-optimal solutions in polynomial time. In practice, near-optimality is usually good enough. An algorithm that returns near-optimal solution is called an approximate algorithm. Note that approximation algorithms are increasingly being used for problems where exact polynomial algorithms are known but are too expensive due to the sizes of the input.

For some approximation algorithms, it is possible to prove certain properties about the approximation of the optimum result. Suppose that we are working on an optimisation problem in which each potential solution has a positive cost, and we wish to find a near-optimal solution. Depending on the problem, an optimal solution may be defined as one with maximum possible cost or one with minimum cost, i.e., the problem may be either a maximisation problem or a minimisation problem.

We say that an algorithm for a problem has an ***approximation ratio*** of ρ, if for any input instance I, the cost $C(I)$ of the solution produced by the algorithm is within a factor of ρ of the cost $C^*(I)$ of an optimal solution:

$$max\left(\frac{C(I)}{C^*(I)}, \frac{C^*(I)}{C(I)}\right) .$$

We also call an algorithm that achieves an approximation ratio of ρ a ***ρ-approximation algorithm***. The definitions of approximation ratio and of ρ-approximation algorithm

apply for both maximisation and minimisation problems. For a maximisation problem, $0 < C(I) \leq C^*(I)$, the ratio $\frac{C^*(I)}{C(I)}$ gives the factor by which the cost of an optimal solution is larger than the cost of the approximate solution. Similarly, for a minimisation problem, $0 < C^*(I) \leq C(I)$, the ratio $\frac{C(I)}{C^*(I)}$ gives the factor by which the cost of the approximate solution is larger than the cost of an optimal solution.

For many problems, polynomial-time approximation algorithms with small constant approximation ratios have been developed. However, for some problems, the best known polynomial-time approximation algorithms have approximation ratios that grow as function of the input size n.

2.3 String problems

In this section, we briefly describe string matching, common subsequence and common supersequence problems.

2.3.1 String matching problem

The ***string matching*** problem is defined as follows. Let $T[0..n-1]$ and $P[0..m-1]$ be strings of length n and m, respectively. Let Σ be a finite alphabet, where Σ may be $\Sigma = \{0, 1\}$ or $\Sigma = \{a, b, \ldots, z\}$ or $\Sigma = \{A, C, G, T\}$ (the DNA alphabet). We further assume that the elements of P and T are characters drawn from Σ.

The string matching problem is the problem to check whether P occurs in T, where an occurrence of P beginning at position i, for $0 \leq i \leq n-1$, in T is defined as $T[i+j] = P[j]$ for all $0 \leq j \leq m-1$. It is known that the string matching problem can be solved in time $O(m+n)$, see e.g., the Knuth-Morris-Pratt algorithm [55] and the variants of Boyer-Moore algorithm [13].

2.3.2 Longest common subsequence (LCS)

Given a sequence $X = x_1 x_2 \ldots x_m$, a *subsequence* of X is a sequence of the form $Y = y_1 y_2 \ldots y_k$ if there exists a strictly increasing sequence $\{i_1, i_2, \ldots, i_k\}$ of in-

dices of X such that for all $j = 1, 2, \ldots, k$, $x_{i_j} = y_j$. For example, $Y = AGGC$ is a subsequence of $X = AGTGACTC$.

Given a set of sequences $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$, a sequence Y is said to be a *common subsequence* of $\mathcal{X}$, if Y is a subsequence of each sequence in $\mathcal{X}$. The ***longest common subsequence*** problem (LCS) is to find the maximum-length common subsequence to all sequences in $\mathcal{X}$. It is known that this problem is NP-hard [64]. When there are only two given sequences with length ℓ in $\mathcal{X}$, the LCS of the two sequences can be found in $O(\ell^2)$ time using straightforward dynamic programming [41]. The time complexity can be further improved by using the $O(\ell^2/\log \ell)$ time dynamic programming [67].

2.3.3 Shortest common supersequence (SCS)

Given two sequences X and Y, X is a *supersequence* of Y if Y can be obtained from X by deleting some (possibly zero) of its elements. Given a set of sequences $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$, a sequence Y is said to be a *common supersequence* of $\mathcal{X}$, if Y is a supersequence of each sequence in $\mathcal{X}$. The ***shortest common supersequence*** problem (SCS) is to find a common supersequence of $\mathcal{X}$ with minimum length. The SCS problem is NP-hard and admits $(n+3)/4$ approximation where n is the number of sequences in $\mathcal{X}$ [30].

For two input sequences X, Y, a SCS can be obtained by finding $LCS(X, Y)$ and inserting into Y the characters in X that are not in $LCS(X, Y)$ while preserving the order in X. The new Y is the SCS. Therefore, $|SCS(X, Y)| = 2\ell - |LCS(X, Y)|$.

2.3.4 Multiple sequence alignment (MSA)

Multiple sequence alignment (MSA) is a sequence alignment of three or more biological sequences that places sequence positions related by function and evolution in the same column of the alignment allowing mismatches and gaps [37]. Roughly

speaking, given a set of $k \geq 2$ sequences, the MSA problem is to align similar subsequences in the same region. Now we give a formal definition of MSA.

Let Σ be a set of characters and $U = \{U_1, U_2, \ldots, U_k\}$ be a set of k sequences, with maximum length m, over Σ. Let $U_i[y]$ denote the y-th character in the sequence U_i for $1 \leq i \leq k$. An alignment of U is a $m' \times k$ matrix

$$U' = \begin{pmatrix} U'_1 \\ U'_2 \\ \vdots \\ U'_k \end{pmatrix}$$

such that $|U'_i| = m'$ and U'_i is formed by inserting spaces into U_i. For a given distance function $\delta(a, b)$ where $a, b \in \Sigma \cup \{-\}$, the *pair-wise score* of U'_i and U'_j is defined as $\sum_{1 \leq y \leq m'} \delta(U'_i[y], U'_j[y])$. A popular assumption in biological alignment is that the score is a *metric*, that is the distance between identical letters is zero and it satisfies the triangular inequality. The *sum-of-pair (SP)* score for an alignment is defined as the sum of the pair-wise scores over all pairs of sequences U'_i and U'_j, i.e.,

$$\text{SP}(U') = \frac{1}{2} \sum_{1 \leq i,j \leq k} \left(\sum_{1 \leq y \leq m'} \delta(U'_i[y], U'_j[y])) \right) .$$

The MSA problem is to find an alignment U' such that $\text{SP}(U')$ is minimised.

For example, we are given three sequences $U_1 = ATCGGC$, $U_2 = CTCGCC$ and $U_3 = TTCGCC$. Suppose

$$\delta(a, b) = \begin{cases} 0 & \text{if } a = b, \\ 1 & \text{if } a \neq b \text{ and } (a = \text{“}-\text{”} \text{ or } b = \text{“}-\text{”}), \\ 2 & \text{otherwise.} \end{cases}$$

. The following alignment has a sum-of-pair score of 18: 4 between U'_1 and U'_2, 8 between U'_1 and U'_3, and 6 between U'_2 and U'_3.

$$\begin{aligned} U'_1 &= \texttt{A-TC-GGC-} \\ U'_2 &= \texttt{-CTC-G-CC} \\ U'_3 &= \texttt{-T-TCG-CC} \end{aligned}$$

Given a weight function $w(i,j)$ for the pair of sequences U_i and U_j, the *weighted sum-of-pair* (SP) score is defined as

$$\mathrm{SP}(U', w) = \frac{1}{2} \sum_{1 \le i,j \le k} w(i,j) \sum_{1 \le y \le m'} \delta(U'_i[y], U'_j[y]) .$$

The ***weighted multiple sequence alignment*** (WMSA) problem is to find an alignment U' such that $\mathrm{SP}(U', w)$ is minimised. WMSA has been proved to be NP-complete. An $O(\log^2 n)$-approximation algorithm [97] has been given via a reduction to the minimum routing cost tree problem (MRCT) [9].

2.4 Elements of graph theory

Graph is a general data structure in computer science. Algorithms for working with graphs are fundamental to the field. There are hundreds of interesting computational problems defined in terms of graphs. In this thesis, we use several definitions and facts from graph theory.

A graph $G = (V, E)$ consists of a collection V of vertices (nodes) and a collection E of edges, each of which joins two of the vertices. Thus, an edge $e \in E$ is represented as a two-element subset of V : $e = (u, v)$ for some $u, v \in V$. *Undirected graph* is that the edges (u, v) and (v, u) are the same. A *directed graph* is a graph such that each edge is an ordered pair (u, v). A *directed acyclic graph* is a directed graph with no cycles.

2.4.1 Tree, binary tree, and binary search tree

An important subclass of graphs are ***trees***. A *rooted tree* is a collection of nodes, one of which is distinguished as a *root*, along with a relation that places a hierarchical structure on the nodes. Formally, a rooted tree is defined recursively in the following manner [3].

1. A single node by itself is a tree. This node is also the root of the tree.

2. Suppose n is a node and $T_1, T_2, \ldots, T_k$ are trees with roots $n_1, n_2, \ldots, n_k$, respectively. We can construct a new tree by making n be the *parent* of nodes $n_1, n_2, \ldots, n_k$. In this tree, n is the root and $T_1, T_2, \ldots, T_k$ are the *subtrees* of the root. Nodes $n_1, n_2, \ldots, n_k$ are called the *children* of node n.

In a tree, nodes without children are called *leaves*. All other nodes are called *internal nodes*. The children of a node are usually ordered from left to right. A special class of trees are ***binary trees***, which is either an empty tree, or a tree in which every node has either no children, a *left child*, a *right child*, or both a left and a right child. A ***binary search tree*** is a binary tree in which each internal node v stores an element such that the elements stored in the left subtree of v are less than or equal to v and elements stored in the right subtree of v are greater than or equal to v. A ***balanced binary search tree*** is a tree that is explicitly kept balanced, e.g., AVL-tree [1]. When there are n nodes in the tree, the height of tree is $\log n$. It is known that the operations of searching, insertion and deletion can be done in time $O(\log n)$ [23] where n is the number of nodes in a balanced binary search tree.

Theorem 1. [23] *The operations searching, insertion and deletion on a balanced binary search tree takes $O(\log n)$ time.*

2.4.2 Minimum routing cost tree (MRCT)

Suppose $G = (V, E)$ is a connected graph in which each edge $(u, v) \in E$ has a weight $c(u, v)$ specifying the cost to connect u and v. A ***spanning tree*** τ for G is a tree containing all nodes in V. The *cost of* τ, denoted by $c(\tau)$, is the sum of the weights of the edges in τ, i.e, $c(\tau) = \sum_{(u,v)\in\tau} c(u, v)$. ***Minimum spanning tree*** is a spanning tree of G such that $c(\tau)$ is minimised.

The *routing cost*, denoted by $c(P(u, v))$, for a pair of vertices (u, v) in a given spanning tree τ is defined as the sum of the weights of the edges in the unique tree path P between u and v, i.e., $c(P(u, v)) = \sum_{(u,v)\in P} c(u, v)$. The *routing cost of* τ, denoted by $rc(\tau)$, is the sum over all pairs of vertices of the routing cost for the

pair in this tree, i.e., $rc(\tau) = \sum_{u,v \in V} c(P(u,v))$ The ***minimum routing cost tree*** (MRCT) is the one with minimum routing cost among all possible spanning trees. Finding a spanning tree of minimum routing cost in a general weighted undirected graph is known to be NP-hard [45]. An $O(\log^2 n)$-approximation algorithm has been given in [9].

Theorem 2. [9] *There is an $O(\log^2 n)$-approximation algorithm for the MRCT problem.*

2.4.3 Travelling salesman problem (TSP)

In the travelling salesman problem (TSP), we are given a complete graph $G = (V, E)$ that has a non-negative integer cost $c(u, v)$ associated with each edge $(u, v) \in E$, we are asked to find a *tour* (a simple cycle that includes all the vertices) of G with minimum cost. The TSP problem is NP-complete [23].

In many practical situations, it is always cheapest to go directly from a place u to a place w; going by way of any intermediate stop v cannot be less expensive. In another word, cutting out an intermediate stop v never increases the cost. We formalise this notation by saying that the cost function c satisfies the ***triangle inequality*** if for all vertices $u, v, w \in V$,

$$c(u, w) \leq c(u, v) + c(v, w) .$$

The TSP problem is NP-complete even if the cost function satisfies the triangle inequality. This means that it is unlikely that an optimal TSP can be found in polynomial time. Therefore, good approximation algorithms are considered. It is known that TSP can be approximated by $3/2$ (Theorem 3).

Theorem 3. [21] *The travelling salesman problem admits a $3/2$-approximation if the weight satisfies the triangle inequality.*

Chapter 3

Randomised Probe Selection Algorithm for Microarray Design

3.1 Introduction

In this chapter, we propose a new approach for the probe selection problem that takes a set of known gene sequences as input and builds a small cardinality set of probes allowing us to identify the unknown target in the sample. Instead of checking all possible probes, randomisation is exploited. We randomly pick probes with some minimal criteria checking. All probes are far (in terms of Hamming distance) from each other. The proposed algorithm performs efficient probe selection and provides unique probes for almost all target sequences in the considered genomes. More detailed discussion on the selection procedure can be found in Section 3.2.3. Our algorithm is quick because exhaustive search is not required. Also, we do not rely on external software.

This chapter is organised as follows. In Section 3.2, the approach for the probe selection problem is presented. The analysis of the time complexity and the experimental results are shown in Section 3.3. The conclusion and discussion of further work are available in Section 3.4.

	TA	AT	GC	CT	GG
ACCTGA	0	0	0	1	0
TGGAT	0	1	0	0	1
CGCGATT	0	1	1	0	0
GTTAC	1	0	0	0	0

Table 3.1: An example of genes and their probes. There is a 1 in an entry g_i and p_j if and only if p_j hybridises to g_i.

3.2 Material and method

In the Section, we first present brief description of the studied probe selection problem, see Section 3.2.1. In Section 3.2.2, we specify the exact criteria used for a probe. In Section 3.2.3, we describe our randomised algorithm. This is followed by the issue of speeding up our algorithm by some combinatorial structure In Section 3.2.4.

3.2.1 Probe selection problem

Given a set of genes $\mathcal{G} = \{g_1, g_2, \ldots, g_n\}$ and a parameter ℓ which specify the length of the probes, the probe selection problem is to find for each gene sequence g_i in $\mathcal{G}$ a length ℓ probe p (that is a substring of g_i) which satisfies the probe selection criteria and belongs only to g_i. If such a probe p does not exist, i.e., p occurs in other sequences in $\mathcal{G}$, then find a small collection of probes that uniquely identifies g.

For example, in Table 3.1, we are interested in finding a unique or a small group of probe(s) for each gene sequence in $\mathcal{G}$. In this example, probe CT is a unique probe of gene sequence ACCTGA, while AT and GG together identify TGGAT.

3.2.2 Probe selection criteria

Every length-ℓ substring of a gene sequence is called a *candidate*. For every candidate, we check whether it satisfies fundamental probe selection criteria: (1) Quantitative criteria; (2) Homogeneity; (3) Sensitivity. Any candidate that passes all these

three criteria is called a *probe*.

Quantitative criteria are described by [62] and are used in Affymetrix probe selection criteria: (1) the content of any single base (As, Ts, Cs or Gs) does not exceed 50% of the candidate size; (2) the length of any contiguous As and Ts or Cs and Gs region is less than 25% of the candidate size; (3) GC-content is between 40% and 60% of the candidate (GC-content is the percentage of nucleotides which are G or C in the sequence).

Homogeneity criterion requires that the melting temperature of candidates should be within some pre-defined range, because a good probe set needs to hybridise to their intended targets at about the same temperature in experiments.

Melting temperature [84] of a probe is the temperature at which 50% of the oligonucleotides and its perfect complement are in duplex. Since it is impossible to know the target DNA concentration, the calculation is approximate, but still useful. Melting temperature T_m of each candidate in our approach is calculated as

$$T_m = \frac{\Delta H}{\Delta S + R \times \ln(c/4)} - 273.15 \tag{3.1}$$

where ΔH and ΔS are the enthalpy and entropy for the helix formation, respectively, R is the molar gas constant (1.987 cal/(K mol)), and c is the total molar concentration of the annealing oligonucleotides when oligonucleotides are not self-complementary. The nearest neighbour model is well adapted to compute the T_m for short sequences, but may lead to an overestimate of the T_m of probes longer than 50nt. Other methods compute T_m by the formula [95] $T_m = 81.5 + (16.6 \log([Na^+])$ + $41[(G + C)/length]$ - $(500/length)$ where $[Na^+]$ is the sodium ion concentration. However, evidence for size limitation of the nearest neighbour model and parameters is sparse [14]. For 70-mer probes, the difference between the T_m values calculated using this method is negligible [96].

Sensitivity criterion filters out candidates prone to self-complementarity (see Figure 3.1). This is to reject all candidates who may fold back on themselves rather

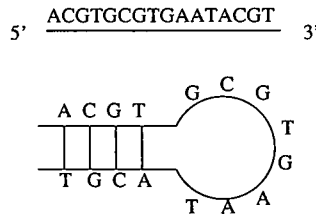

Figure 3.1: A candidate prone to self-complementarity.

than on target sequences. Consider every segment of a candidate of length x. If its reversal forms a consecutive length x complementary segment within itself, the candidate is considered prone to fold back on itself.

Another useful measure for sensitivity is the free energy. The total difference in the free energy of the folded and unfolded states of a DNA duplex is approximated by a nearest-neighbour model:

$$\Delta G_i(total) = \sum_j n_{ij}\Delta G_j + \Delta G_i(init) + \Delta G_i(sym) \tag{3.2}$$

where each different oligonucleotide duplex is given the subscript i, ΔG_j is the free energy for the 10 possible Watson-Crick nearest-neighbour stacking interactions, n_{ij} is the number of occurrences of each nearest neighbour j, in each sequence i, $\Delta G_i(init)$ is the initiation free energy, and $\Delta G_i(sym)$ equals +0.4 kcal/mol if duplex i is self-complementary and zero if it is non-self-complementary [16]. DNA oligonucleotide nearest-neighbour thermodynamic parameters are available [85] and they allow prediction of oligonucleotide DNA hybridisation energies.

The thermodynamic parameters used in our melting temperature and free energy calculation were estimated from experimental measurements on short probes. Therefore, although we used both to model long probe binding stability, the free energy values should be viewed as a function of binding stability on a relative scale, rather than be interpreted as the absolute free energy generated during DNA duplex formation.

In this work, we are mainly interested in efficient selection of *unique probes*, playing a role of gene signatures. We say that probe p is a *unique probe* for gene g in a genome if and only if p occurs in g and there is no close occurrence (in terms of Hamming distance, see Specificity criterion) of p in any other gene of the genome.

Specificity identifies probes that are unique to each gene in the genome. This condition minimises cross-hybridisation of the probes with other gene sequences. Hamming distance has been used as the basis for coding theoretic approaches [31, 60] to the DNA word design problem. In particular, Hamming distance becomes a powerful tool for determining closeness/similarity and recently has been adopted as the specificity measure [59, 78, 90]. Thus, if the Hamming distance between a probe and every candidate (excluding those candidates from the gene where the probe belongs to) is greater than some constant, the probe is said to be specific enough [1].

3.2.3 Randomised probe selection algorithm

In this section, a new algorithm to select probes for DNA microarrays is presented. Initially, the algorithm exploits several filters (based on probe selection criteria) to reduce the search space for probes. However, the main idea used here is to exploit randomisation to reduce the time complexity of the search. And indeed, randomly generated sequences are expected to possess properties of unique probes. E.g., probe selection criteria enforce balanced distribution of base pairs in probes which is naturally satisfied by random sequences. Moreover, the Hamming distance between two randomly chosen sequences of length ℓ over a 4 letter alphabet is about $3\ell/4$, which is also highly desired property of a system of probes.

The proposed probe selection algorithm starts with the filtering stage applied on the whole genome. For each candidate, we test whether it passes the probe selection criteria (1), (2) and (3) and the candidates which fail the test are eliminated. For

[1] Our approach is independent from any particular specificity criterion (whether Hamming distance or BLAST search) is used. Our algorithm can be adopted any other specificity criteria as a black box.

Algorithm 1 Probe selection.

Input: A set of gene sequences $\mathcal{G}$, the specified length of probes ℓ and the Hamming distance threshold d with default value 5.

Output: A set of probes $\mathcal{P}$.

Steps:

1: $i \Leftarrow 0$ and $not\_found \Leftarrow true$;
2: **for** every gene $g \in \mathcal{G}$: **do**
3: **while** $i < 5$ and not_found is $true$ **do**
4: generate a random sequence r_i of length ℓ;
5: find the closest probe p_i in gene g;
6: **if** $H(p_i, q) \geq d$ for all candidates q in other genes in $\mathcal{G} - \{g\}$ **then**
7: p_i is chosen as the unique probe for g, report p_i, $not\_found \Leftarrow false$;
8: **end if**
9: $i \Leftarrow i + 1$;
10: **end while**
11: **end for**

(2) Homogeneity, the melting temperature is within the range $[78, 90]$; for (3) Sensitivity, the candidates with a self-complementary segment of length greater than or equal to 4 are rejected.

When the filtering is completed, we iterate a probe selection procedure which acts on all genes in the genome. The probe selection procedure, see Algorithm 1, runs with gene $g \in \mathcal{G}$, generates a unique (if it is able to find it) probe p for gene g. T his is done as follows: (a) generate a random sequence r of length ℓ; (b) find the closest match p to r among probes in the target; (c) check whether p satisfies specificity criterion. This process is iterated at most five times which allows us to obtain a good trade-off between the accuracy of the search procedure and its running time. We have fixed the number of iterations to five times by testing the performance against the number of iterations. We observed that the percentage of targets identified by a single probe becomes stable after five iterations (see Figure 3.2). The code of the procedure could be easily modified to incorporate the case when a unique probe is not found, in this case, we check whether a combination of any two (and very rarely three) already selected probes uniquely identifies the considered gene g.

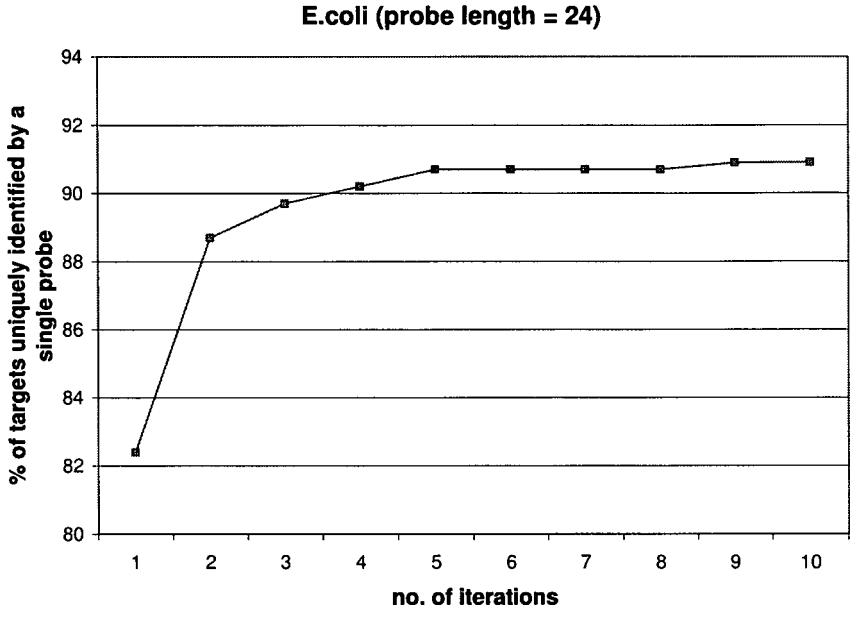

Figure 3.2: Percentage of targets identified by a single probe becomes stable after five iterations.

It should be pointed out that the proposed algorithm terminates once probes have been found to satisfy the probe selection criteria, rather than searching for optimal probes. In this end, we are in line with [14, 78, 82, 83, 90, 92, 96]. Using this strategy, the algorithm can select probes for large genomes for which algorithms demanding optimality are unsuccessful [46, 59].

3.2.4 Speeding up methods

To speed up the probe selection procedure, we exploit an "encoding" method to test self-complementarity and specificity. Consider every segment of a candidate of length 4, if its reversal forms complementary segment within itself, the candidate is prone to form a secondary structure. In particular, every segment of a candidate of

length ℓ is encoded as follows:

$$\sum_{i=0}^{\ell-1} c_i \times 4^{(\ell-i-1)} \tag{3.3}$$

where c_i is either 0, 1, 2 or 3 (standing for A, C, G, T, respectively) representing the ith base of the segment. For example, a sequence ATCG is encoded as $0 \times 4^3 + 3 \times 4^2 + 1 \times 4^1 + 2 \times 4^0 = 54$. Furthermore, we exploit the tabling method to speed up the specificity checking process. We pre-compute a matrix $D = [D_{ij}]$ in which the rows and columns are indexed by numerical values obtained (by Formula 3.3) from all possible DNA sequences of length 4. Each entry D_{ij} is the Hamming distance between two DNA sequences with numerical value i and j. For example, if $i = 0$, representing AAAA, and $j = 255$, representing TTTT, then $D_{0,255} = 4$. By looking up the appropriate entry in the table, Hamming distance between two probes of length-ℓ can be quickly determined.

3.3 Implementation and results

The experimental results show that the proposed algorithm is much faster than existing algorithms especially for large genomes. For more commonly tested datasets, Table 3.2 summaries the relative performance of the proposed algorithm with some algorithms mentioned in Section 1.3.1. The randomised procedure selects probes efficiently from short (24 bases) through long (64 bases) probes for large genomes. Furthermore, the proposed approach significantly reduces the number of probes needed in microarray design.

The length of the probes designed by existing software ranges from 20 to 70: around 20 [46, 59, 62, 90, 92], around 30 [46, 78] around 50 [59, 83, 90], and around 70 [14, 59, 90, 96]. The developed software is able to design probes of various length in this range (see Section 3.3.2).

As for the number of probes returned, some algorithms returned all probes [90] requiring longer computational time while most of the other software return a small

	Li and Stormo, 2001	Rahmann, 2002	the proposed algorithm
E.coli	23nt, 1.5 days	24nt, 32 minutes	64nt, 20 minutes
S.cerevisiae	24nt, 4 days	24nt, 116 minutes	64nt, 40 minutes
N.crassa	more than a week	24nt, 240 minutes	24nt, 155 minutes
Human chromosome 1	a few weeks	space exhausted	64nt, 740 minutes

Table 3.2: Comparison of the proposed algorithm and other algorithms.

number of probes. We follow the approach adopted by most software and report a small number.

3.3.1 Time complexity analysis

The brute force approach for specificity checking scans through the whole length-n genome for every length-ℓ probe and determines if the Hamming distances are large enough. Such a process is computationally expensive, requiring $O(\ell n^2)$ time. In comparison, we pick up a probe of length ℓ by using randomisation for every gene in the genome, then scan through the whole genome for specificity checking. By doing this, we do not need to check every probe in each gene which greatly reduce the time complexity. Thus, the time complexity of the proposed algorithm is $O(k\ell n)$ where k is the number of genes in the whole genome, ℓ is the length of probe and n is the length of the whole genome. Usually k is much smaller than n, e.g., in Human chromosome 1, the value of k is 2,017 while n is 197,317,844.

3.3.2 Analysis of experimental results

The developed software RANDPS is written in C and is developed and tested on Athlon XP2000+ Cluster with 2GB memory. The software is available on the following website (http://www.csc. liv.ac.uk/~cindy/RandPS/RandPS.htm). The size of RANDPS code is 25KB which is simple and clean while being efficient and effective. Inputs of RANDPS are FASTA formatted gene sequences, downloaded from the NCBI website (http://www. ncbi.nlm.nih.gov/). RANDPS uses a size-n array,

	Total length of genes	No. of genes	Avg. length per gene	Time (minutes)
E.coli	4,752,411	5,253	905	20
S.cerevisiae	8,783,280	5,888	1,492	40
S.pombe	7,272,320	5,471	1,329	60
N.crassa	17,484,362	10,633	1,644	310
A.thaliana	33,581,216	26,186	1,282	1520
Mouse chromosome 2	182,887,278	1,302	140,466	470
Human chromosome 1	197,317,844	2017	97,827	740

Table 3.3: Information of the datasets and time used for RANDPS of probe length 64.

where n is the concatenated length of gene sequences of a genome, to store the inputs, together with another two size-n arrays to store the corresponding numerical value of each base in the genome and the status (*candidate* or *probe*) of each position in the concatenated sequence.

The experiments were undertaken in order to evaluate the performance of the developed software on various types of genomes. We report the results using several genomes that have been widely used for the probe selection problem. These datasets have been used in experiments in [46, 59, 78, 83, 90, 92]. In terms of time consumption, for probe length 64, it takes about 20 minutes to process the E.coli genome, 40 minutes to process the S.cerevisiae genome, 60 minutes for S.pombe, 310 minutes for N.crassa, 470 minutes for Mouse chromosome 2, about 740 minutes for Human chromosome 1 and 1520 minutes for A.thaliana. The genomes involved in the experiments and corresponding time used are listed in Table 3.3.

In terms of accuracy of probe selection, we are able to find unique probes for up to 99% of genes in the whole genome. The full details of the experimental results are shown in Tables 3.4-3.10 [2]. We have run experiments 30 times on each dataset

[2]The melting temperature range has been slightly modified for longer probe lengths 48, 56, and 64.

Genome	E.coli		
Length	4,752,411		
# of genes	5,253		
Probe length	Number of genes requiring		
	1 probe	2 probes	no probe returned
24	4759 (90.7%)	490 (9.3%)	4
32	4791 (91.3%)	457 (8.7%)	5
40	4805 (91.6%)	442 (8.4%)	6
48	4808 (91.7%)	436 (8.3%)	9
56	4827 (92.1%)	413 (7.9%)	13
64	4832 (92.3%)	405 (7.7%)	16

Table 3.4: Results of RANDPS for E.coli.

for each probe length. In these tables, the first three rows are basic information about the datasets, which are the name of the genome, the length of the genome and the number of genes in the genome. The column "Probe length" lists the different lengths we used to test the performance of the developed software. The column "1 probe" shows the number of genes which can be identified by a unique probe, while "2 probes" column shows the number of genes which require a combination of two probes for unique identification. The percentages in brackets are calculated on the basis of the number of genes with probes (i.e., total number of genes minus number of genes without probes). The "no probe returned" column shows the number of genes where the developed software did not find feasible probes.

The experimental results in Table 3.4 show that RANDPS is able to find a unique probe for over 90% of E.coli with different probe lengths. The remaining genes can be identified by a combination of two probes. There are only around 10 genes where the proposed algorithm did not find feasible probes. For other genomes with similar number of genes (S.cerevisiae and S.pombe), around 95% genes can be identified by using a single probe. The results can be found in Tables 3.5 and 3.6. Tables 3.7 and 3.8 illustrate that for genomes with larger number of genes (N.crassa

Genome	S.cerevisiae		
Length	8,783,280		
# of genes	5,888		
Probe length	Number of genes requiring		
	1 probe	2 probes	no probe returned
24	5481 (93.2%)	401 (6.8%)	6
32	5516 (93.9%)	361 (6.1%)	11
40	5525 (94.2%)	341 (5.8%)	22
48	5549 (94.7%)	313 (5.3%)	26
56	5560 (95.0%)	292 (5.0%)	36
64	5560 (95.1%)	288 (4.9%)	40

Table 3.5: Results of RANDPS for S.cerevisiae.

and A.thaliana), up to 99% genes can be identified by one probe. Finally, for larger datasets of length over 180M (Mouse chromosome 2 and Human chromosome 1), results are shown in Tables 3.9 and 3.10. In this case, RANDPS is able to select unique probes for over 95% of the datasets.

In the experiments, we have noticed that there are some genes with no probe. An investigation of these genes revealed that some of these genes are duplicated or very similar to some other genes in the genome. Another reason is that the lengths of some of these genes are too short. Apart from these cases, the developed software is able to select probes for all genes.

As further illustration of the developed software in terms of accuracy of the probe set, we compare the free energy of a group of the probes selected by RANDPS with the optimal probes with minimum free energy, which is found by using a brute force approach. This is shown in Figures 3.3-3.9 on samples of one hundred arbitrarily chosen genes for each genome. A closer look into the mean and standard deviation (Table 3.11) of hybridisation free energy between the optimal probes and the probes chosen by RANDPS reveals that the probes we found are very close to the optimal one. Thus the developed software is able to find high quality probes.

Genome	S.pombe		
Length	7,272,320		
# of genes	5,471		
Probe length	Number of genes requiring		
	1 probe	2 probes	no probe returned
24	5061 (92.6%)	407 (7.4%)	3
32	5064 (92.6%)	404 (7.4%)	3
40	5131 (94.1%)	321 (5.9%)	19
48	5141 (94.3%)	308 (5.7%)	22
56	5154 (94.6%)	294 (5.4%)	23
64	5152 (94.7%)	287 (5.3%)	32

Table 3.6: Results of RANDPS for S.pombe.

3.4 Discussion

We have proposed a new approach to select (randomly) a small set of probes and demonstrated that such a small set of probes is sufficient to distinguish each gene from all the other genes in the genome. Almost all genes can be identified by a unique probe, the others need at most two probes. We have implemented a probe selection software RANDPS, which runs efficiently. The software is available on line at http://www.csc.liv. ac.uk/~cindy/RandPS/RandPS.htm.

We believe that the proposed approach should prove to be useful also in the design of multiple probes. Multiple probes might be needed for several reasons. E.g., to accommodate a lack of accuracy in experimental work, a fault-tolerant system is desirable. In some experimental situations, the mRNA is broken into random fragments, which thus require multiple probes per gene.

Therefore, one of the future direction would be on identification and classification of genes by multiple probes. This requires adaptation of the proposed algorithm. We expect the running time to increase, yet this is worthwhile for the scenario we described above.

Genome	N.crassa		
Length	17,484,362		
# of genes	10,633		
Probe length	Number of genes requiring		
	1 probe	2 probes	no probe returned
24	10530 (99.2%)	90 (0.8%)	13
32	10551 (99.5%)	57 (0.5%)	25
40	10557 (99.5%)	50 (0.5%)	26
48	10558 (99.6%)	45 (0.4%)	30
56	10559 (99.6%)	42 (0.4%)	32
64	10544 (99.6%)	40 (0.4%)	49

Table 3.7: Results of RANDPS for N.crassa.

In future research, it would be interesting to improve performance of the proposed algorithm on more complex organisms, since the structure of higher organism differs from that of bacteria and viruses. This would lead to a more challenging combinatorial problem.

Another direction would be further studies on sensitivity. There have been several improvements in the calculation of minimum free energy in recent software UNAFOLD [66]. Although UNAFOLD is not yet used directly into probe selection, it is important to consider UNAFOLD in probe selection as future work.

Genome	A.thaliana		
Length	33,581,216		
# of genes	26,186		
Probe length	Number of genes requiring		
	1 probe	2 probes	no probe returned
24	22407 (85.6%)	3773 (14.4%)	6
32	24400 (93.2%)	1777 (6.8%)	9
40	24813 (94.8%)	1358 (5.2%)	15
48	25094 (95.9%)	1063 (4.1%)	29
56	25238 (96.5%)	910 (3.5%)	38
64	25327 (96.9%)	807 (3.1%)	52

Table 3.8: Results of RANDPS for A.thaliana.

Genome	Mouse chromosome 2		
Length	182,887,278		
# of genes	1,302		
Probe length	Number of genes requiring		
	1 probe	2 probes	no probe returned
24	1194 (91.7%)	108 (8.3%)	0
32	1229 (94.4%)	73 (5.6%)	0
40	1231 (94.5%)	71 (5.5%)	0
48	1235 (94.9%)	67 (5.1%)	0
56	1239 (95.2%)	63 (4.8%)	0
64	1240 (95.2%)	62 (4.8%)	0

Table 3.9: Results of RANDPS for Mouse chromosome 2.

Genome	Human chromosome 1		
Length	197,317,844		
# of genes	2,017		
Probe length	Number of genes requiring		
	1 probe	2 probes	no probe returned
24	1718 (85.2%)	299 (14.8%)	0
32	1914 (94.9%)	103 (5.1%)	0
40	1918 (95.1%)	99 (4.9%)	0
48	1926 (95.5%)	91 (4.5%)	0
56	1931 (95.7%)	86 (4.3%)	0
64	1932 (95.8%)	85 (4.2%)	0

Table 3.10: Results of RANDPS for Human chromosome 1.

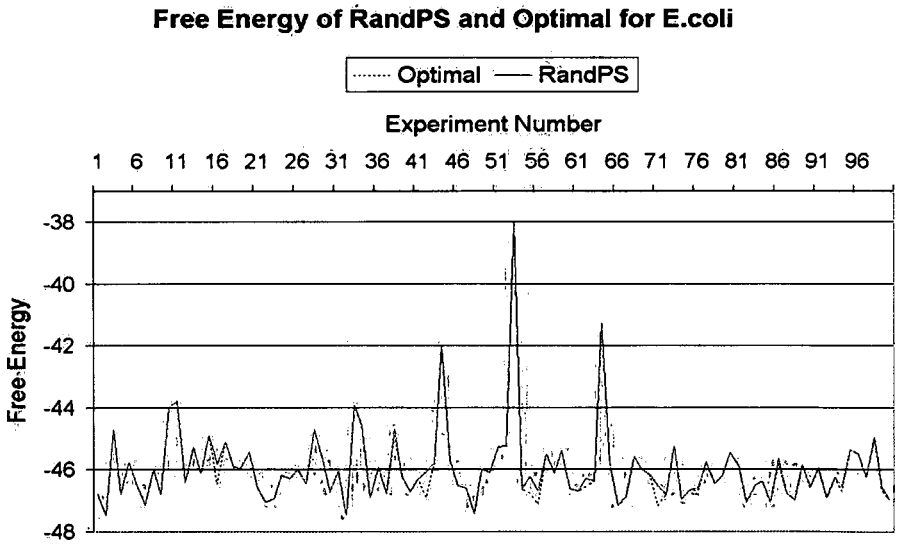

Figure 3.3: Comparison of free energy between the optimal probe and the probe chosen by RANDPS for E.coli.

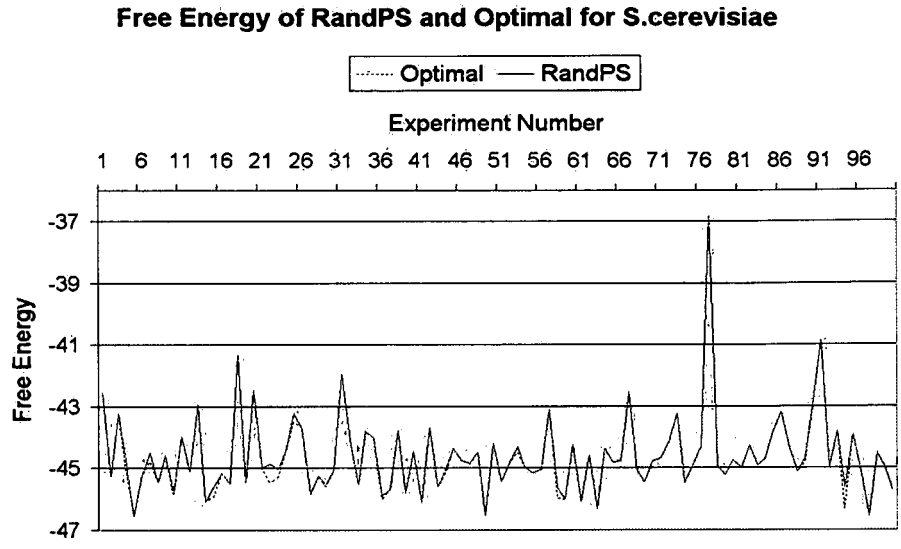

Figure 3.4: Comparison of free energy between the optimal probe and the probe chosen by RANDPS for S.cerevisiae.

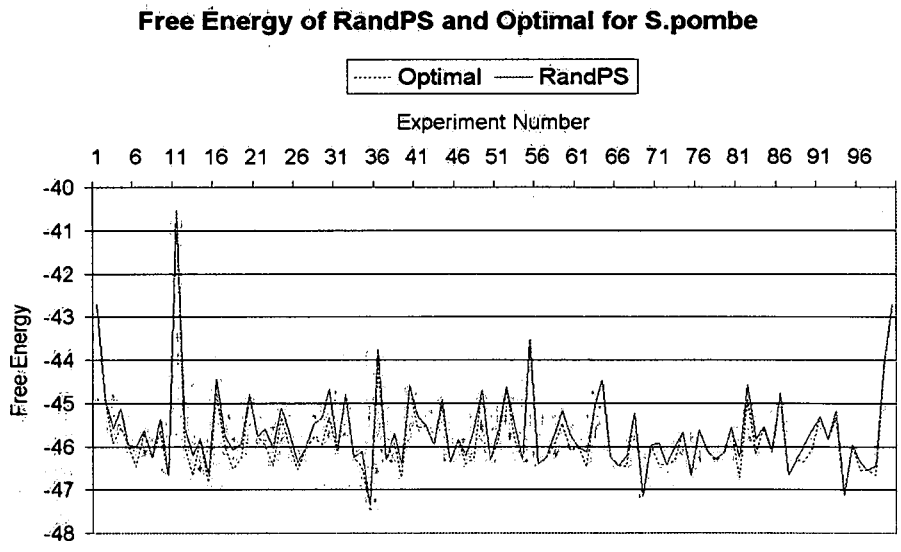

Figure 3.5: Comparison of free energy between the optimal probe and the probe chosen by RANDPS for S.pombe.

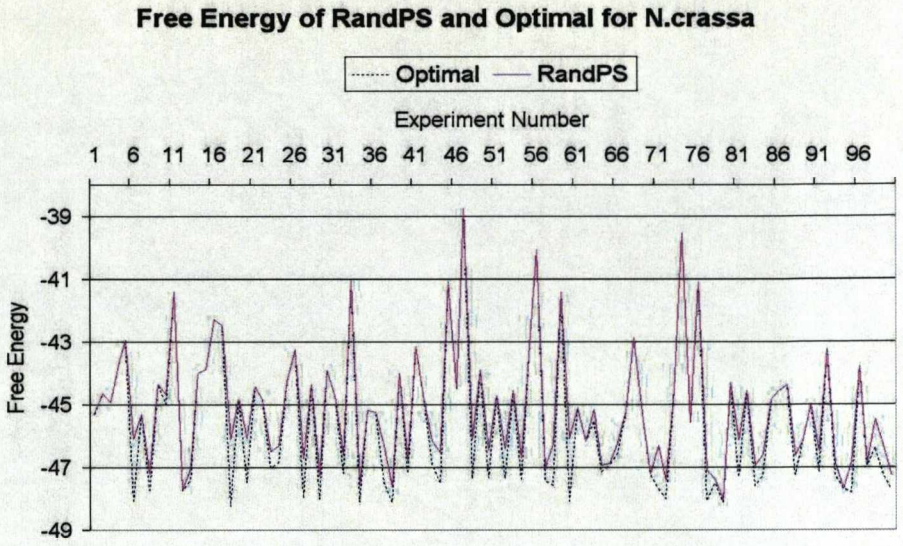

Figure 3.6: Comparison of free energy between the optimal probe and the probe chosen by RANDPS for N.crassa.

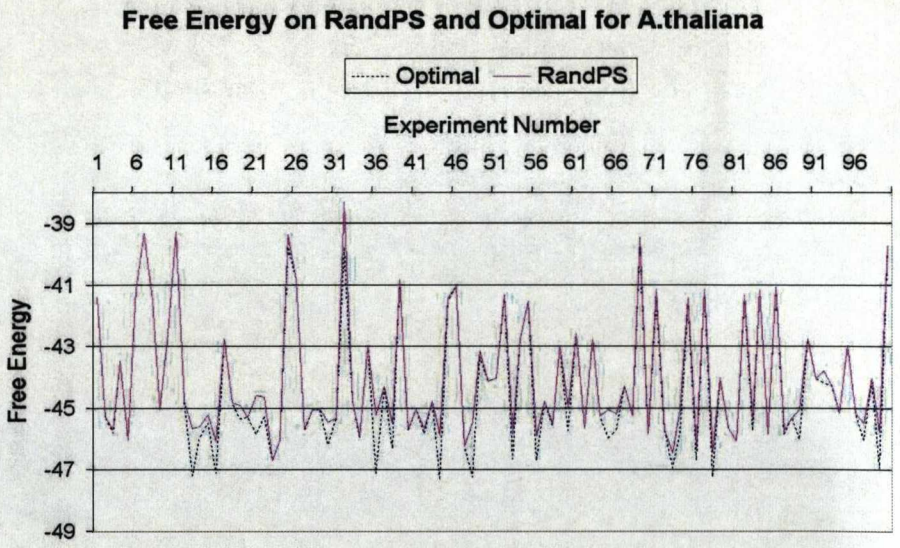

Figure 3.7: Comparison of free energy between the optimal probe and the probe chosen by RANDPS for A.thaliana.

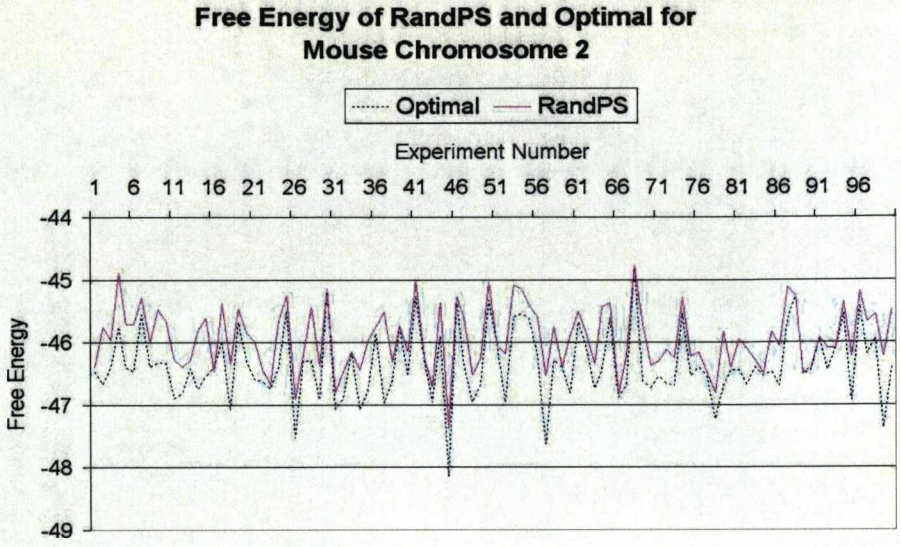

Figure 3.8: Comparison of free energy between the optimal probe and the probe chosen by RANDPS for Mouse chromosome 2.

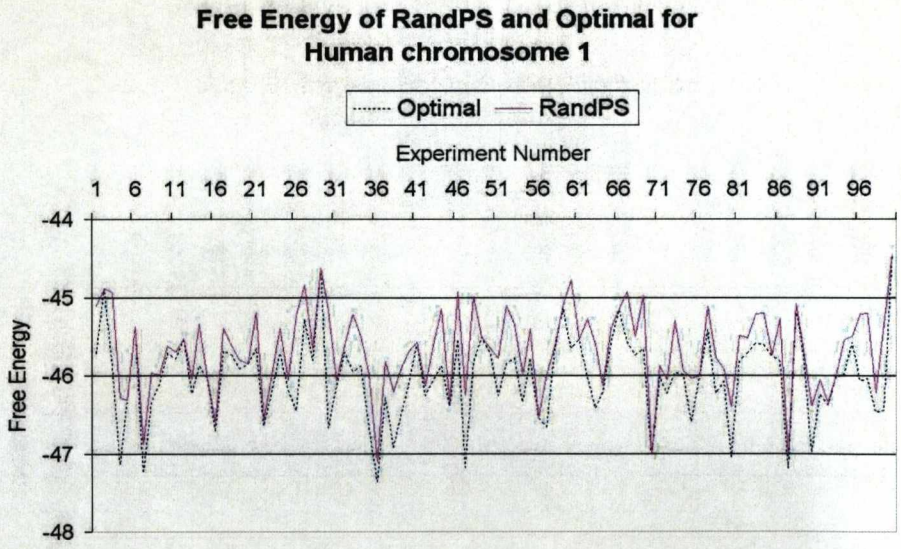

Figure 3.9: Comparison of free energy between the optimal probe and the probe chosen by RANDPS for Human chromosome 1.

	OPT	RANDPS	Absolute Difference
E.coli	-45.991 (1.288)	-45.949 (1.272)	0.042 (0.016)
S.cerevisiae	-44.625 (1.347)	-44.586 (1.329)	0.039 (0.018)
S.pombe	-45.989 (0.863)	-45.609 (0.805)	0.380 (0.058)
N.crassa	-45.490 (2.196)	-45.149 (1.915)	0.341 (0.281)
A.thaliana	-44.439 (2.200)	-44.098 (2.025)	0.341 (0.175)
Mouse chromosome 2	-46.363 (0.579)	-45.937 (0.516)	0.426 (0.063)
Human chromosome 1	-46.020 (0.544)	-45.676 (0.534)	0.344 (0.010)

Table 3.11: Comparison of mean and standard deviation of free energy between the optimal probe and the probe chosen by RANDPS. The values are represent by mean (standard deviation).

Chapter 4

Faster Algorithm for the set variant of the String Barcoding Problem

4.1 Introduction

In this chapter, we improve the time complexity $O(n^2|\mathcal{S}|)$ of the approximation algorithm for the test set problem proposed by Berman *et al.* [10], which solves also the variant of the string barcoding problem adopted here. The proposed algorithm works in almost optimal time $O(n|\mathcal{S}|\log^3 n)$ in view of the fact that the size of the input to the studied problem is of size $\Omega(n|\mathcal{S}|)$ (the input is very often expressed as the binary matrix with $|\mathcal{S}|$ rows and n columns, where the entry (i,j) set to 1 (0) means that substring j belongs (does not belong) to string i, see Table 4.1). The improved time complexity is a direct consequence of more careful management of processed sets, use of several specialised graph and string data structures as well as tighter time complexity analysis based on the amortised argument.

The chapter is organised as follows. We present first a short description of the studied variant of the string barcoding problem and basic notation and definitions used later in the chapter, see Section 4.2.1. In Section 4.2.2, we highlight implementation differences between the proposed algorithm and its counterpart from [10]. This is followed by a detailed description of specialised data structures used by the proposed algorithm, see Section 4.2.3. The analysis of the time complexity based on

	AC	C	GA
$AGGT$	0	0	0
$ACCTGA$	1	1	1
$TGGAT$	0	0	1
GCA	0	1	0
$CGCGATT$	0	1	1
$GTTAC$	1	1	0

Table 4.1: An example of genes and their probes. There is a 1 in an entry g_i and p_j if and only if p_j hybridises to g_i.

an amortised argument is presented in Section 4.2.4. The conclusion and discussion of further work are available in Section 4.2.3.

4.2 The method to solve string barcoding problem

We start this section with a short introduction to the (limited) variant of the string barcoding problem.

4.2.1 String barcoding problem

Given a set of $|\mathcal{G}|$ gene sequences (targets), $\mathcal{G} = \{g_1, g_2, ..., g_{|\mathcal{G}|}\}$. The objective is to find as small as possible set of elements (probes) $\mathcal{P} = \{p_1, p_2, ..., p_m\}$ from a given set of substrings of strings in $\mathcal{G}$ with cardinality n, such that, for any pair of strings $g_i, g_j \in \mathcal{G}$, there is at least one probe $p \in \mathcal{P}$ that is a substring of g_i or g_j, but not two of them. We say $\mathcal{P}$ distinguishes $\mathcal{G}$ if this property holds. The hybridisation pattern can be viewed as a string of m zeros and ones, referred to as the barcode of each target sequence in $\mathcal{G}$.

For example, let $\mathcal{G}$ be the set of input strings $\{AGGT, ACCTGA, TGGAT, GCA, CGCGATT, GTTAC\}$. Then, the set $\mathcal{P} = \{AC, C, GA\}$ with cardinality 3 gives the set of valid barcodes (shown in Table 4.1) for the input sequences in $\mathcal{G}$.

For simplicity of presentation, we assume here that n is a power of two. Other-

wise, the set of n elements can be extended by some dummy elements. We use $[i..j]$ to denote the set of consecutive integers $\{i, i+1, ..., j-1, j\}$ and $P(T)$ to denote the power set of a set $T \subseteq [0..n-1]$. The complement $[0..n-1] \setminus T$ of T is denoted by $\overline{T}$. The cardinality of a set T is represented by $|T|$. We say that a set T *distinguishes* two elements $x, y \in [0..n-1]$ where $x \neq y$, if $|\{x, y\} \cap T| = 1$.

Definition 4. *(Test set problem TS)*
INSTANCE: $(n, \mathcal{S})$ *where* $\mathcal{S} \subseteq P([0..n-1])$.
VALID SOLUTION: *A collection* $\mathcal{T} = \{T_0, T_1, ..., T_{|\mathcal{T}|-1}\} \subseteq \mathcal{S}$ *such that for every pair of distinct integers* $x, y \in [0..n-1]$, *there exists* $T \in \mathcal{T}$ *that distinguishes* x *and* y.
OBJECTIVE: *minimise* $|\mathcal{T}|$.

Example 1: Let $n = 3$ and $\mathcal{S} = \{\{0\}, \{1\}, \{0, 1\}\}$. Then, $\mathcal{T} = \{\{0\}, \{0, 1\}\}$ is a valid solution since $\{0, 1\}$ distinguishes 0 from 2 ($|\{0, 2\} \cap \{0, 1\}| = 1$) as well as 1 from 2 ($|\{1, 2\} \cap \{0, 1\}| = 1$) while $\{0\}$ distinguishes 0 from 1 ($|\{0, 1\} \cap \{0\}| = 1$).

Definition 5. *(Equivalence relation $\overset{\mathcal{T}}{\equiv}$)*
Assume that a collection $\mathcal{T}$ *is a valid solution to the instance* $(n, \mathcal{S})$ *of the test set problem. An equivalence relation* $\overset{\mathcal{T}}{\equiv}$ *is defined on* $[0..n-1]$ *and the collection* $\mathcal{T}$, *where for any* $i, j \in [0..n-1]$, $i \overset{\mathcal{T}}{\equiv} j$ *if and only if* $\forall T \in \mathcal{T}$, *either* $\{i, j\} \subseteq T$ *or* $\{i, j\} \cap T = \emptyset$. *The equivalence relation* $\overset{\mathcal{T}}{\equiv}$ *partitions* $[0..n-1]$ *into* m *equivalence classes, where the* l^{th} *equivalence class is denoted by* $E(\mathcal{T}, l)$, *for* $l = 0, 1, ..., m-1$ *and let* $E(\mathcal{T}) = \{E(\mathcal{T}, 0), E(\mathcal{T}, 1), ..., E(\mathcal{T}, m-1)\}$.

Note that each non-trivial equivalence class $E(\mathcal{T}, l)$ of $\overset{\mathcal{T}}{\equiv}$ is a product of the form $T_0^* \cap T_1^* ... \cap T_{|\mathcal{T}|-1}^*$, where $T_k^* = T_k$ or $T_k^* = \overline{T_k}$, for $k = 0, 1, ..., |\mathcal{T}| - 1$.

Definition 6. *(Entropy function $H_{\mathcal{T}}$)*
The entropy function $H_{\mathcal{T}}$ *is defined as* $H_{\mathcal{T}} = \log_2(\Pi_{l=0}^{m-1} |E(\mathcal{T}, l)|!)$ *where* m *is the*

number of equivalence classes in $E(\mathcal{T})$.

Example 2: Let $\mathcal{T} = \{T_0\}$, where $T_0 = \{2,3,4\}$ and $n = 8$. Then, $E(\mathcal{T},0) = T_0 = \{2,3,4\}$, $E(\mathcal{T},1) = \overline{T_0} = \{0,1,5,6,7\}$. So, $E(\mathcal{T}) = \{\{2,3,4\},\{0,1,5,6,7\}\}$ and $H_{\mathcal{T}} = \log_2((3!)(5!)) \approx 9.492$.

Example 3: Let $\mathcal{T} = \{T_0, T_1\}$ where $T_0 = \{2,3,4\}$, $T_1 = \{1,3,5,7\}$ and $n = 8$. Then, $E(\mathcal{T},0) = T_0 \cap T_1 = \{3\}$, $E(\mathcal{T},1) = T_0 \cap \overline{T_1} = \{2,4\}$, $E(\mathcal{T},2) = \overline{T_0} \cap T_1 = \{1,5,7\}$, $E(\mathcal{T},3) = \overline{T_0} \cap \overline{T_1} = \{0,6\}$. As a result, $E(\mathcal{T}) = \{\{3\},\{2,4\},\{1,5,7\},\{0,6\}\}$ and $H_{\mathcal{T}} = \log_2((1!)(2!)(3!)(2!)) \approx 4.585$.

Definition 7. *(Combination of equivalence relations* $\otimes$*)*
A combination of two equivalence relations $\overset{\mathcal{T}}{\equiv}$ *and* $\overset{\mathcal{T}'}{\equiv}$ *on the set* $[0..n-1]$ *is defined as* $E(\mathcal{T}) \otimes E(\mathcal{T}') = E(\mathcal{T} \cup \mathcal{T}')$.

Definition 8. *(Basic block)*
A basic block $B(i,l)$ *is a set of consecutive integers* $[(i-1) \times n/2^l..i \times n/2^l - 1]$, *where* $1 \leq i \leq 2^l$ *and* $0 \leq l \leq \log n$.

4.2.2 String barcoding algorithms

In this chapter, we propose a more efficient implementation of the algorithm due to Berman, DasGupta and Kao [10] for the TS problem.

Algorithm 2 Berman, DasGupta and Kao [10]

```
1: T = ∅;
2: while H_T ≠ 0 do
3:     select a T_j ∈ S \ T that minimises H_{T∪{T_j}};
4:     T = T ∪ {T_j};
5: end while
```

As mentioned earlier, Berman *et al.* [10] proposed an $O(n^2|\mathcal{S}|)$ time approximation algorithm for TS with the approximation ratio $(1 + \ln n)$. In each round of their algorithm, where rounds correspond to consecutive iterations of loop while in Algorithm 2, they compute combinations $E(\mathcal{T}) \otimes E(\{T_j\})$ for all $T_j \in \mathcal{S} \setminus \mathcal{T}$. They also select T_j that minimises the entropy function $H_{\mathcal{T} \cup \{T_j\}}$ and then move T_j (from $\mathcal{S} \setminus \mathcal{T}$) to the collection $\mathcal{T}$. Since for each remaining T_j, a naive computation of $E(\mathcal{T}) \otimes E(\{T_j\})$ and $H_{\mathcal{T} \cup \{T_j\}}$ takes time $\Omega(n)$, and since for most rounds, $|\mathcal{S} \setminus \mathcal{T}| = \Omega(|\mathcal{S}|)$, each round requires time $\Omega(n|\mathcal{S}|)$. Algorithm 2 is executed in at most $n - 1$ rounds because n integers can be separated by at most $n - 1$ sets from $\mathcal{S}$ in the worst case. Therefore, the total complexity of Algorithm 2 is $O(n^2|\mathcal{S}|)$. Note also that in Algorithm 2, no information about $E(\mathcal{T}) \otimes E(\{T_j\})$ and the entropy $H_{\mathcal{T} \cup \{T_j\}}$ is kept for future use in later rounds (apart from T_j that minimises the entropy function).

Algorithm 3 Amortised algorithm

1: $\mathcal{T}(0) = \emptyset$;
2: **for** $j = 1, 2, ..., |\mathcal{S}|$ **do**
3: Compute $E(\{[0..n-1]\} \cup \{T_j\})$;
4: **end for**
5: $T(0) = T_j$ such that T_j minimises $H_{\{T_j\}}$;
6: **for** $(i = 1; H_{\mathcal{T}(i-1)} \neq 0; i++)$ **do**
7: /* i is the number of current round */
8: $\mathcal{T}(i) = \mathcal{T}(i-1) \cup \{T(i-1)\}$;
9: **for** $j = 1, 2, ..., |\mathcal{S}|$ **do**
10: Compute $E(\mathcal{T}(i) \cup \{T_j\})$ by applying $E(\mathcal{T}(i-1) \cup \{T_j\}) \otimes E(\mathcal{T}(i-1) \cup \{T(i-1)\})$;
11: **end for**
12: $T(i) = T_j$ such that T_j minimises $H_{\mathcal{T}(i) \cup \{T_j\}}$;
13: **end for**
14: **return** $\mathcal{T}(i-1)$;

In this chapter, the main idea is to store and to utilise information about all previously computed combinations $E(\mathcal{T}) \otimes E(\{T_j\})$ together with the history of their entropies computation (see Algorithm 3). This is to reduce the overall time

complexity. Let $T(i-1)$ be the selected set that minimises the entropy function at the end of round $i-1$. Let $\mathcal{T}(i) = \{T(0), T(1), ..., T(i-1)\}$ represent the collection of sets selected as part of the solution in rounds $0, 1, ..., i$. Since we keep records on all combinations $E(\mathcal{T}(i-1)) \otimes E(\{T_j\})$ obtained in round $i-1$, later during round i, we can compute $E(\mathcal{T}(i) \cup \{T_j\})$ applying $E(\mathcal{T}(i-1) \cup \{T_j\}) \otimes E(\mathcal{T}(i-1) \cup \{T(i-1)\})$ rather than via direct computation of $E(\mathcal{T}(i) \cup \{T_j\})$ as it is done in Berman *et al.* algorithm. We introduce a new concept of hierarchical data structure (see Section 4.2.3) that allows to represent and manipulate equivalence classes $E(\mathcal{T}(i) \cup \{T_j\})$ and $E(\mathcal{T}(i) \cup \{T(i-1)\})$ efficiently. Moreover, we make use of a directed acyclic graph to compute the history of the entropies (see Section 4.2.3).

4.2.3 Implementation details: analysis of data structures

We introduce a hierarchical data structure $\mathcal{H}$ to represent, compare and process efficiently a dynamic collection of sets $\mathcal{C}$ of small integers, i.e., subsets of $[0..n-1]$. Initially $\mathcal{C} = \mathcal{S}$, and later it contains all (including intermediate) subsets of $[0..n-1]$ corresponding to all considered equivalence classes generated by Algorithm 3. The new data structure allows equality tests on two sets from the collection to be performed in constant time. Moreover, single element insertions to and deletions from any set from the collection are implemented in poly-logarithmic time.

Binary tree representation of a set

In principle, each set S in $\mathcal{C}$ is represented by a binary tree structure D_s (of pointers) defined as follows. In each tree, there are exactly $\log n + 1$ levels enumerated from 0 (root level) to $\log n$ (leaf level). At the level l there are 2^l nodes. Each internal node v in the tree is the parent of two children, the left child $l(v)$ and the right child $r(v)$. Moreover, each node of the tree representing S stores information about the content of S projected on a specific basic block, chosen according to the following rule. The root of the tree stores information about the content of S projected on $B(1,0)$.

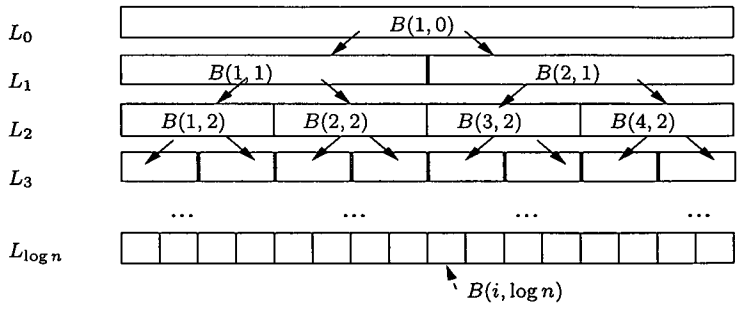

Figure 4.1: Binary tree representation of a set.

And later if a parent node v on level l stores the information about the content of S projected on $B(i,l)$, then $l(v)$ and $r(v)$ store information about the contents of S projected on $B(2i-1,l+1)$ and $B(2i,l+1)$ respectively. For example, the leaves store information about the content of set S projected on consecutive basic blocks $B(1,\log n)$, $B(2,\log n)$, ..., $B(i,\log n)$, ..., $B(n,\log n)$ which are either empty sets or singletons (see Figure 4.1).

Hierarchical data structure for a collection of sets

The nodes of binary tree structures representing sets from the collection $\mathcal{C}$ whose contents refer to the same basic block *correspond* to each other and we say that they belong to the same *group*. The binary tree structures representing sets in $\mathcal{C}$ are stored in the hierarchical data structure $\mathcal{H}$ in a compact form, where two corresponding nodes (associated with the same basic block) in different trees with the same content (the same subset of $[0..n-1]$) are represented by a single node in $\mathcal{H}$ (see Figure 4.2). In order to create $\mathcal{H}$ (from the trees) and further manipulate it efficiently, we introduce a variant of the naming method (see e.g. [52]).

Naming method The naming method adopted here requires application of a system of *counters* and *balanced binary search trees*. Each group of nodes based on a specific basic block $B(i,l)$, for $0 \leq i \leq n-1$ and $0 \leq l \leq \log n$, requires a separate counter $C_{i,l}$ (that is used to generate new names within the group of nodes) and a

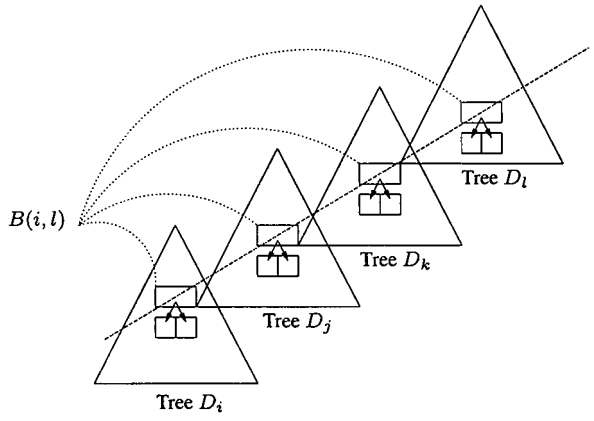

Figure 4.2: Hierarchical data structure for a collection of sets.

balanced binary search tree $T_{i,l}$ (that keeps all used names in the group of nodes indexed by the pair of children's names). In each tree of the collection embedded in $\mathcal{H}$, the nodes get integer names, level by level, starting from the lowest level $\log n$.

The counters are initialised to value 0 and the balanced binary search trees are set to be empty. At the bottom level ($\log n$), the i-th leaf, for any $0 \leq i \leq n-1$, in the tree representing S (embedded in $\mathcal{H}$) is given name 0 if $\{i\} \cap S = \emptyset$ and 1 otherwise. Above that, at each consecutive higher level $1 \leq l \leq \log n$ in every tree D_s, for all $S \in \mathcal{C}$, and at every internal node $v \in D_s$ associated with some $B(i,l)$, we first check whether the pair of names $(N(l(v)), N(r(v)))$ already occurs as the pair of names of children of some corresponding node w in some other tree $D_{s'}$. This can be done in time $O(\log n)$ by searching for the pair $(N(l(v)), N(r(v)))$ as the key in the balanced binary search tree $T_{i,l}$. If the pair $(N(l(v)), N(r(v)))$ does not occur as the key in $T_{i,l}$, we increase the counter $C_{i,l}$ by 1 and assign its new value as the name of v. Moreover we insert to $T_{i,l}$ a new record with the content $N(v)$ and the key $(N(l(v)), N(r(v)))$. This is also done in time $O(\log n)$. Otherwise, if there already exists a node w such that $(N(l(v)), N(r(v))) = (N(l(w)), N(r(w)))$, v adopts the name of w, i.e., $N(v) = N(w)$, and v is represented by the node w in $\mathcal{H}$.

Lemma 9. *The initialisation of the hierarchical structure $\mathcal{H}$ is done in time $O(n|\mathcal{S}|\log n)$.*

Proof. The nodes in the hierarchical structure $\mathcal{H}$ get integer names , level by level, starting from the lowest level $\log n$. As explained above, the computation of a single name including manipulation of respective data structure in $\mathcal{H}$ requires time $O(\log n)$. At each level l for $0 \leq l \leq \log n$, there are at most $2^l|\mathcal{S}|$ nodes to be named. Thus, to generate names of nodes at level l in $\mathcal{H}$ requires time $O(2^l|\mathcal{S}|\log n)$. In conclusion, the initial computation of the names of all nodes in $\mathcal{H}$ is done in time $O(\sum_{l=0}^{l=\log n} 2^l|\mathcal{S}|\log n)$, which is $O(n|\mathcal{S}|\log n)$. □

Set operations

In the amortised analysis argument provided in section 4.2.4, we use three operations performed on sets from the dynamic collection $\mathcal{C}$. Namely, *equality test* $Eq(S, S')$ for the contents of two sets $S, S' \in \mathcal{C}$, i.e., whether $S = S'$, *deletion operation* $Delete(S, x)$ that removes x from S, i.e., $S = S \setminus \{x\}$ and *insertion operation* $Insert(S, x)$ that adds x to S, i.e., $S = S \cup \{x\}$. When we perform equality test $Eq(S, S')$ on two sets from $\mathcal{C}$, we only need to compare the names of nodes representing sets S and S' in $\mathcal{H}$. This can be done in constant time. When we remove an element x from a set S ($Delete(S, x)$), we change the name from 1 to 0 of the appropriate node v representing x in S located at the bottom level in $\mathcal{H}$ and then we update the names of all nodes on the path from the node v to the node representing the whole set S at the top level of $\mathcal{H}$. Since there are $O(\log n)$ names to be changed at different levels in $\mathcal{H}$ and the computation of the name of a node in $\mathcal{H}$ requires time $O(\log n)$ as explained in Section 4.2.3, the deletion operation takes time $O(\log^2 n)$. The insertion operation ($Insert(S, x)$) is implemented analogously (where we change name from 0 to 1 at the bottom level of $\mathcal{H}$) to the deletion operation. As a result, we get the following lemma.

Lemma 10. *The structure $\mathcal{H}$ provides a mechanism for $O(1)$-time equality test for two sets in $\mathcal{C}$ and $O(\log^2 n)$-time single element removal from and insertion to a set in $\mathcal{C}$.*

Efficient cross-examination of equivalence classes

Note that in any advanced round i of Algorithm 3 each equivalence relation $E(\mathcal{T}(i-1) \cup \{T_j\})$ may potentially have $\Omega(n)$ equivalence classes. Thus a naive cross-examination with all classes in $E(\mathcal{T}(i-1) \cup \{T(i-1)\})$ (see line 10 in Algorithm 3) may lead to $\Omega(|\mathcal{S}|n)$ comparisons during each round. And since the number of rounds may be as large as $\min(|\mathcal{S}|, n)$ we would see no improvement in the time complexity in comparison with the algorithm presented in [10].

In order to reduce the number of cross-examined equivalence classes we provide another data structure SL (structured list of equivalence classes) based on unique names of classes available in the hierarchical structure $\mathcal{H}$ and defined as follows. Assume that during round i we have an equivalence relation $E(\mathcal{T}(i-1))$ formed of c equivalence classes $E_1 = E(\mathcal{T}(i-1), 1), \ldots, E_c = E(\mathcal{T}(i-1), c)$, s.t., each class $E_x = E(\mathcal{T}(i-1), x)$ is potentially split into two classes E_x^L and E_x^R (possibly empty) in $E(\mathcal{T}(i)) = E(\mathcal{T}(i-1) \cup \{T(i-1)\})$ (see Figure 4.3). Also each equivalence relation $E(\mathcal{T}(i-1) \cup \{T_j\})$ potentially bears two subclasses $E_x^{L(j)}$ and $E_x^{R(j)}$ (possibly empty) for each $E(\mathcal{T}(i-1), x) \in E(\mathcal{T}(i-1))$. We assume that at the beginning of round i the structure SL is formed of c lists $L_1, L_2, .., L_c$, s.t., each L_x associated with E_x contains all different pairs of subclasses (multiplicities are

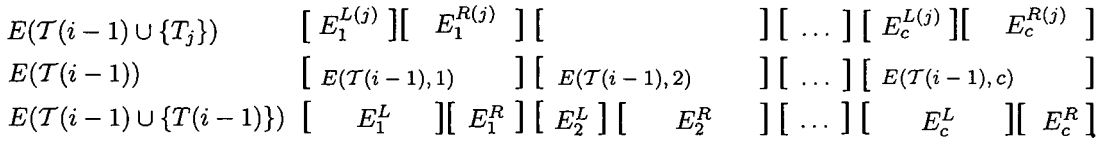

Figure 4.3: Cross-examination of equivalence classes.

discarded to avoid dummy cross-examinations) of E_x present both in $E(\mathcal{T}(i-1) \cup \{T(i-1)\})$ and in each $E(\mathcal{T}(i-1) \cup \{T_j\})$. On the conclusion of round i each list L_x in SL is split (if needed) into two lists associated with two equivalence classes E_x^L and E_x^R, where each of these lists contains now all pairs of different subclasses in new $E(\mathcal{T}(i) \cup \{T_j\})$. Finally note that since every cross-examination of two different pairs of sub-classes results in creation of new equivalence classes (at least one split) the number of all cross-examined pairs can be bounded by $O(|\mathcal{S}| \cdot n)$. The total cost of all handling the structured list of equivalence classes has to be multiplied by the factor of $O(\log^2 n)$ which refers to access to and location of new equivalence classes in the hierarchical structure $\mathcal{H}$. This results in the total complexity $O(|S| n \log^2 n)$.

Entropy function calculation

The entropy function is computed dynamically on the basis of a directed acyclic graph $\vec{G}$ (see Figure 4.4) gradually expanded during consecutive rounds of Algorithm 3. We keep at each node in $\vec{G}$ the name and the size of the equivalence class it represents. At the end of round i, for $0 \leq i \leq n-1$, all values of the entropy function $H_{\mathcal{T}(i) \cup \{T_j\}}$, for all $T_j \in \mathcal{S} \setminus \mathcal{T}(i)$, are calculated. The set T_j which minimises $H_{\mathcal{T}(i) \cup \{T_j\}}$ is selected as $T(i)$. The use of $\vec{G}$ allows to reduce the overall cost (on the top of handling the hierarchical structure $\mathcal{H}$) of computation of the entropy function to $O(|\mathcal{S}|n)$. We prove later that this cost is linear in the total number of *splits* of equivalence classes $E(\mathcal{T}(i) \cup \{T_j\})$ represented by nodes in $\mathcal{H}$ through out consecutive rounds of Algorithm 3.

Recall that in a directed graph, nodes without successors are called *sinks*, and nodes with no predecessors are called *source* nodes. The acyclic graph $\vec{G}$ is created and maintained as follows. At the top level of $\vec{G}$, see Figure 4.4, we place $|\mathcal{S}|$ nodes labelled by R_j, for $1 \leq j \leq |\mathcal{S}|$, where each node represents the whole range $[0..n-1]$ before any of $T_j \in \mathcal{S}$ is introduced. These nodes will be the only sinks in $\vec{G}$ throughout the duration of the algorithm. In round 0, each set R_j is partitioned into T_j and $\overline{T_j}$ (the complement of T_j). The nodes labelled by the

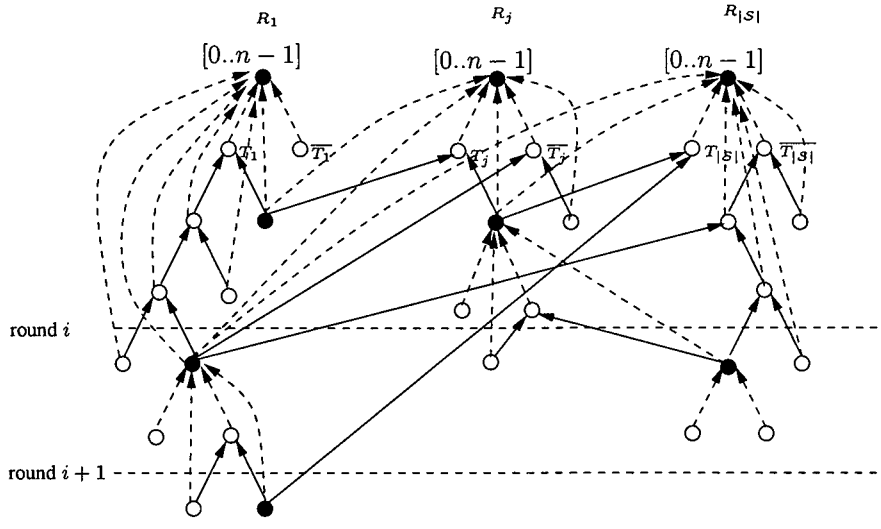

Figure 4.4: Compute entropy function by using $\vec{G}$. Express link is shown by dash line.

names of T_j and $\overline{T_j}$ become temporary sources. They are inserted into $\vec{G}$ as the predecessors of the sink labelled by R_j. The entropy function $H_{\{R_j\}\cup\{T_j\}} = H_{\{T_j\}}$ is calculated directly on the basis of information available in newly generated nodes (the sizes of T_j and $\overline{T_j}$) and its value $H_{\{T_j\}}$ is stored at the sink labelled by R_j, for $1 \leq j \leq |\mathcal{S}|$. The set T_j which minimises $H_{\{T_j\}}$ is selected as $T(0)$.

Later, at the beginning of round i, each source node in $\vec{G}$ is labelled by the name of some equivalence class $\mathcal{E} \in E(\mathcal{T}(i-1)\cup\{T_j\})$ represented by some node in $\mathcal{H}$. We also have $T(i-1)$ which is calculated during round $i-1$. Note that if $\mathcal{E} \not\subseteq T(i-1)$ and $\mathcal{E} \cap T(i-1) \neq \emptyset$ (intersection of $\mathcal{E}$ and $T(i-1)$ is non-trivial), the source node labelled by the name of $\mathcal{E}$ becomes a successor of two new nodes. We also say that $\mathcal{E}$ is split. The two new nodes are labelled by the names of two new equivalence classes $\mathcal{E} \cap T(i-1), \mathcal{E} \cap \overline{T(i-1)} \in E(\mathcal{T}(i)) \otimes E(\{T_j\})$. If any two newly obtained equivalence classes $\mathcal{E}_j \in E(\mathcal{T}(i)\cup\{T_j\})$ and $\mathcal{E}_{j'} \in E(\mathcal{T}(i)\cup\{T_{j'}\})$, for $j \neq j'$, have the same content, they are represented by the same node in $\vec{G}$ called a *branching node*. We colour all branching nodes as well as all sinks to black, see

Figure 4.4. All other nodes in $\vec{G}$ remain white. Moreover, we create a collection of *express links* such that every (black or white) node v is connected via express link to the first black nodes w on a directed path leading to any R_j reachable from v. The following lemma holds.

Lemma 11. *A structure of all nodes connected via express links from any node v in $\vec{G}$ forms a tree rooted in v with all R_js reachable from v as leaves, where the number of leaves subsumes the number of internal nodes.*

Proof. By construction, every node v is connected via express link to the first black nodes w which can either be a branching node or a sink. Moreover, the directed express path rooted from v will finally reach some sink labelled by R_j. This holds due to the set represented by v must be a subset of some range $[0..n-1]$ which is a sink in $\vec{G}$. Therefore, the structure of all nodes connected via express links from any node v forms a tree where v is the root and all R_js reachable from v are the leaves. Assume that a node v in $\vec{G}$ is connected to x sinks which are the leaves in the spanning tree of v. There are at most $x-1$ branching nodes in the spanning tree of v. □

The value $H_{T(i)\cup\{T_j\}}$ computed in round i only needs to be updated when a temporary source v connected to R_j is split into two new nodes. Let the size of v be s and the sizes of the two new nodes be s_1 and s_2, respectively. Recall the definition of the entropy function, when a split happens, $H_{T(i)\cup\{T_j\}} = \frac{f(s_1)f(s_2)}{f(s)} H_{T(i)}$. The fraction $\frac{f(s_1)f(s_2)}{f(s)}$ can be delivered to the sink via the spanning tree of v. In such a way, $H_{T(i)\cup\{T_j\}}$ can be calculated efficiently and the set T_j which minimises $H_{T(i)\cup\{T_j\}}$ is selected as $T(i)$.

Lemma 12. *Let $C(i)$ be the number of splits of equivalence classes in round i. The maintenance cost (on the top of manipulation of $\mathcal{H}$) of $\vec{G}$ in round i is $O(C(i))$.*

Proof. Assume that a node v (corresponding to some equivalence class) in $\vec{G}$ is connected to x (the number equivalence relations containing v as a class) sinks.

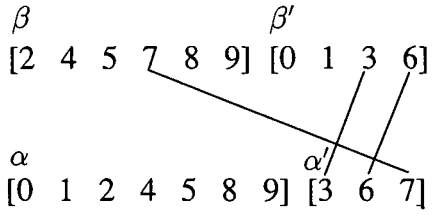

Figure 4.5: Two different pairs of equivalence classes.

When v gets split, x sinks have to be informed and updated. By Lemma 11, this is done in time $O(x)$ due to the presence of express links in the spanning tree spanned on at most $x-1$ branching nodes. Therefore, the maintenance cost of $\vec{G}$ in round i is $O(C(i))$. □

Corollary 13. *Since the total number of splits in all sets R_j is no more than $|\mathcal{S}|(n-1)$ the on-line maintenance of all current values of the entropy function is done at cost $O(|\mathcal{S}|n)$.*

4.2.4 Amortised analysis

In this section we show that the total cost of the proposed string barcoding procedure is $O(n|\mathcal{S}|\log^3 n)$.

Assume that during round i two different pairs of equivalence classes (α,α') and (β,β'), where $\alpha = E_x^L$ and $\alpha' = E_x^R$ form a split of a class $E_x = E(\mathcal{T}(i-1),x)$ (in $E(\mathcal{T}(i-1))$ caused by $T(i-1)$ and $\beta = E_x^{L(j)}$ and $\beta = E_x^{R(j)}$ form a split of the same class E_x caused by $T(j)$, are available in the list L_x (see Figure 4.5). As the result of cross-examination we obtain new four sets $\alpha\beta = \alpha \cap \beta$, $\alpha\beta' = \alpha \cap \beta'$ $\alpha'\beta = \alpha' \cap \beta$, and $\alpha'\beta' = \alpha' \cap \beta'$ that form two new pairs of equivalence classes $(\alpha\beta,\alpha\beta')$ and $(\alpha'\beta,\alpha'\beta')$ to be considered during the next round. Assume also that $|\alpha'| \leq |\alpha|$ (note that information about the sizes of α and α' can be either kept in the hierarchical structure $\mathcal{H}$ or it could be computed on line from the size of a superclass formerly split into α and α'). The proposed algorithm takes all elements

(one by one) from α' and searches for their occurrences in β and β'. When an element is located in β it is moved to the set $\alpha'\beta$ otherwise it is moved from β' to $\alpha'\beta'$. When this process is finished whatever is left in β becomes $\alpha\beta$ and whatever remains in β' becomes $\alpha\beta'$. The split operation is completed.

In the amortised analysis argument we would like to trade in tested elements from α' for the total cost of the proposed string barcoding procedure. Thus in round i the equivalence classes in all $E(\mathcal{T}(i) \cup T_j)$ overlapping with E_x will be updated at the uniform cost $|\alpha'|$, for each T_j outside of $\mathcal{T}(i)$. And this is happening only when α' is non-empty, otherwise no cost is charged (there will be no pair (α, α') in L_x) since there will be no immediate split of classes β and β' in $E(\mathcal{T}(i-1) \cup T_j)$. Also when $(\alpha, \alpha') = (\beta, \beta')$ no split is required, and indeed in this case both pairs appear as one in L_x. Since we always charge the cost of a split to a smaller set α' (i.e., $|\alpha'| \leq |\alpha|$) every element in each $E(\mathcal{T}(i-1) \cup T_j)$ will be charged at most $\log n$ times during the whole execution of Algorithm 3. Note also that the search in β and β' for each charged element takes time $O(\log^2 n)$. This is done with a help of the hierarchical structure $\mathcal{H}$ and the procedures $Delete()$ and $Insert()$, see section 4.2.3. This means that the total charge across all ($|\mathcal{S}|$) equivalence relations $E(\mathcal{T}(i-1) \cup T_j)$ can be limited to $O(|\mathcal{S}|n \log^3 n)$.

Theorem 14. *Algorithm 3 is $O(|\mathcal{S}|n \log^3 n)-$ time string barcoding approximation procedure with the approximation ratio $O(1 + \log n)$.*

Proof. The time complexity follows from the amortised argument presented above. The $O(1 + \log n)$ approximation ratio is provided by the algorithm presented in [10], i.e., the proposed string barcoding procedure does not change sets selected to the final solution. We just show how to perform the selection process in a more efficient way. □

4.3 Discussion

In this chapter, we improved on the time complexity $O(n^2|\mathcal{S}|)$ of the approximation algorithm for the test set problem proposed by Berman *et al.* [10], which solves also the variant of the string barcoding problem adopted here. The proposed algorithm works in almost optimal time $O(n|\mathcal{S}|\log^3 n)$ in view of the fact that the size of the input to the studied problem is of size $\Omega(n|S|)$.

Among problems to be still addressed in the context of this work is efficient design of fault-tolerant barcodes in which every pair of strings are separated by two (or more) probes available in the pool of precomputed probes.

Chapter 5

Approximating Border Length for DNA Array Synthesis

In this chapter, approximation of the border minimisation problem (BMP) in asynchronous synthesis is studied which is believed to be NP-hard. As far as we know, this is the first result with proved performance guarantee. The main result is an $O(\sqrt{n}\log^2 n)$-approximation, where n is the number of probes in the microarray. This is based on an approximation algorithm for the variant when the placement of probes is given in advance (called P-BMP problem) and we are asked to find the embeddings to minimise the total border length. We show that P-BMP is $O(\log^2 n)$-approximable. We further show that if the array is one-dimensional, P-BMP can be solved optimally in polynomial time and there is a constant approximation for BMP. On the other hand, we show that BMP can be defined as the maximum agreement problem (MAP) with a different objective called "agreement". Minimising the border length is equivalent to maximising the agreement. Yet we are able to devise $O(1)$-approximation algorithms for MAP regardless of whether the placement is given in advance or not.

Technically speaking, the result is based on a reduction of BMP problem to the weighted multiple sequence alignment problem (WMSA) [65, 97]. We employ an approximation algorithm for WMSA, which is based on a result for the minimum routing cost spanning tree problem (MRCT) [9, 20].

Organisation of the chapter. In Section 5.1, definitions and notation are given. In Sections 5.2 and 5.3, we present and analyse approximation algorithms for BMP and MAP, respectively. Finally we give a conclusion and discuss some future work in Section 5.4.

5.1 Introduction

We are given a set of n length-ℓ probes $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$. For any sequence p_i, we denote the t-th character of a sequence p_i by $p_i[t]$. The probes in $\mathcal{P}$ are to be placed on an $\sqrt{n} \times \sqrt{n}$ array (for simplicity, we assume that $\sqrt{n}$ is an integer). This array is represented by a grid graph $G = (V, E)$. Two grid vertices (x_1, y_1) and (x_2, y_2) are said to be *neighbour* if $|x_1 - x_2| + |y_1 - y_2| = 1$. For each vertex $v \in V$, we denote the set of neighbours of v by $\mathcal{N}(v)$.

Placement and embedding. A *placement* of the probes is a bijective function $\phi : \mathcal{P} \to V$ that maps each probe to a unique vertex in the grid G. An *embedding* of a set of probes $\mathcal{P}$ into a deposition sequence D is denoted by $\varepsilon = \{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n\}$. For $1 \leq i \leq n$, ε_i is a length-$|D|$ sequence such that (1) $\varepsilon_i[t]$ is either $D[t]$ or a space " $-$ "; and (2) removing all spaces from ε_i gives p_i. The hamming distance between ε_i and ε_j measures the border length between p_i and p_j if they are neighbours in a certain placement. We define this quantity as the *conflict* between the embeddings of p_i and p_j, denoted by $\mathrm{conf}_\varepsilon(p_i, p_j)$. Note that $\mathrm{conf}_\varepsilon(p_i, p_j) \leq 2\ell$. We define the *share* between the embeddings of p_i and p_j as $2\ell - \mathrm{conf}_\varepsilon(p_i, p_j)$, and denote it by $\mathrm{share}_\varepsilon(p_i, p_j)$.

Border length and agreement. The *border length* of a placement ϕ and an embedding ε is defined as the sum of conflicts between the embeddings of probes that are neighbours in the placement ϕ in G:

$$\mathrm{BL}(\phi, \varepsilon) = \frac{1}{2} \sum_{\substack{p_i, p_j : \\ \phi(p_j) \in \mathcal{N}(\phi(p_i))}} \mathrm{conf}_\varepsilon(p_i, p_j)\,. \tag{5.1}$$

The BMP problem is to find a placement ϕ and an embedding ε, so that $\mathrm{BL}(\phi, \varepsilon)$ is minimised. We denote the optimal placement and the corresponding optimal embedding by ϕ^* and ε^*, respectively. We further define the counter part of border length, the *agreement*, which is the sum of shares between the embeddings of probes that are neighbours in the placement ϕ in G:

$$\mathrm{A}(\phi, \varepsilon) = \frac{1}{2} \sum_{\substack{p_i, p_j : \\ \phi(p_j) \in N(\phi(p_i))}} \mathrm{share}_\varepsilon(p_i, p_j) \tag{5.2}$$

The *Maximum Agreement Problem* (MAP) is to find a placement ϕ and an embedding ε, so that $\mathrm{A}(\phi, \varepsilon)$ is maximised. Since $\mathrm{A}(\phi, \varepsilon) = 4\ell(n - \sqrt{n}) - \mathrm{BL}(\phi, \varepsilon)$, minimising the border length $\mathrm{BL}(\phi, \varepsilon)$ is equivalent to maximising the agreement $\mathrm{A}(\phi, \varepsilon)$.

Common subsequence and common supersequence. The border length is closely related to the common subsequence and common supersequence between neighbouring sequences in the placement. Consider any two length-ℓ sequences p, q, we denote the longest common subsequence and shortest common supersequence of two sequences p and q by $LCS(p, q)$ and $SCS(p, q)$, respectively, and the corresponding length as $|LCS(p, q)|$ and $|SCS(p, q)|$, respectively. $SCS(p, q)$ can be obtained by finding $LCS(p, q)$ and inserting into p the characters in q that are not in $LCS(p, q)$ while preserving the order in q. Therefore, $|SCS(p, q)| = 2\ell - |LCS(p, q)|$. For any embedding ε, the maximum number of common deposition nucleotides between p and q is $|LCS(p, q)|$, in other words, $\mathrm{conf}_\varepsilon(p, q) \geq 2(\ell - |LCS(p, q)|)$ and $\mathrm{share}_\varepsilon(p, q) \leq 2|LCS(p, q)|$. We define the *LCS distance* to be $2(\ell - |LCS(p, q)|)$, denoted by $\mathrm{dist}(p, q)$. In other words, $\mathrm{dist}(p, q)$ is a lower bound of $\mathrm{conf}_\varepsilon(p, q)$ for any embedding ε.

Multiple sequence alignment (MSA) and Weighted MSA (WMSA). As we will see in later sections, a variant of BMP problem, named P-BMP (BMP problem in which the placement is given), can be polynomial time reducible to WMSA. As a consequence, we can apply the approximation results on WMSA to P-BMP,

which can be further used as a building block for the approximation for BMP. We first review the MSA and WMSA problems. MSA and WMSA are computational biology problems that have been studied extensively [6, 11, 28, 36, 81]. Let Σ be a set of characters and $U = \{U_1, U_2, \ldots, U_k\}$ be a set of k sequences, with maximum length m, over Σ. An alignment of U is a $m' \times k$ matrix

$$U' = \begin{pmatrix} U'_1 \\ U'_2 \\ \vdots \\ U'_k \end{pmatrix}$$

such that $|U'_i| = m'$ and U'_i is formed by inserting spaces into U_i. For a given distance function $\delta(a, b)$ where $a, b \in \Sigma \cup \{-\}$, the *pair-wise score* of U'_i and U'_j is defined as $\sum_{1 \le y \le m'} \delta(U'_i[y], U'_j[y])$. Given a weight function $w(i, j)$ for the pair of sequences U_i and U_j, the *weighted sum-of-pair* (SP) score

$$\text{SP}(U', w) = \frac{1}{2} \sum_{1 \le i,j \le k} w(i, j) \sum_{1 \le y \le m'} \delta(U'_i[y], U'_j[y]) .$$

The WMSA problem is to find an alignment U' such that $\text{SP}(U', w)$ is minimised. WMSA has been proved to be NP-complete. An $O(\log^2 n)$-approximation algorithm [97] has been given via a reduction to the minimum routing cost tree problem (MRCT) [9].

Minimum routing cost tree problem (MRCT). In this problem, a graph with weighted edges is given. For a spanning tree of the graph, the *routing cost* between two vertices is the sum of weights of the edges on the unique path between the two vertices in the spanning tree. The *routing cost* of the spanning tree is defined as the sum of routing cost between every pair of two vertices. The MRCT problem is to find a spanning tree whose routing cost is minimum. The results in [9] state that there is a polynomial time reduction from WMSA to MRCT. Each sequence in the input of WMSA corresponds to a vertex in the input graph of MRCT. The edge weight between two vertices is set to be the weighted edit distance between the two corresponding sequences. The reduction result states that (1) there is a routing

spanning tree τ whose routing cost is at most $O(\log^2 n)$ times $\sum_{i,j} w(i,j)d(i,j)$, where $d(i,j)$ is the edit distance between the two sequences i and j; and (2) there is an alignment U' whose SP(U', w) is at most the routing cost of τ. Note that $\sum_{i,j} w(i,j)d(i,j)$ is a lower bound on the weighted SP score. Therefore, the following lemma follows.

Lemma 15. [97] *There is an $O(\log^2 n)$-approximation algorithm for the WMSA problem, where n is the number of sequences to be aligned.*

5.2 The border minimisation problem (BMP)

In this section, we study the BMP problem. In general, we are to find a placement and an embedding for the given probe set. An $O(\sqrt{n}\log^2 n)$-approximation algorithm is given for BMP (Section 5.2.2), which is based on an approximability result for a variant of BMP, named P-BMP (Section 5.2.1). At the end of this section, the case when the array is one-dimensional is also discussed, and we show that BMP admits better results in this case.

5.2.1 Approximation for P-BMP

We first study the P-BMP problem, a variant of BMP with a placement given in advance. The concern becomes to find an embedding. We show that P-BMP is $O(\log^2 n)$-approximable by giving a reduction of P-BMP to the weighted multiple sequence alignment problem (WMSA), for which there is an $O(\log^2 n)$-approximation algorithm [97].

Lemma 16. *There is a polynomial time reduction from P-BMP to WMSA.*

Proof. Let I be an instance of the P-BMP problem with a given placement ϕ. We construct an instance I' for WMSA such that there is a solution for I with border length X if and only if there is a solution for I' with a weighted SP score of X.

Construction of I'. We first show the construction of I'. The input sequence for WMSA is the same as the input probe set $\mathcal{P}$. The weight $w(i,j)$ is defined as follows:

$$w(i,j) = \begin{cases} 1 & \text{if } \phi(p_j) \in \mathcal{N}(\phi(p_i)), \\ 0 & \text{otherwise.} \end{cases}$$

The distance function $\delta(a,b)$, for $a, b \in \Sigma \cup \{-\}$, is defined as follows:

$$\delta(a,b) = \begin{cases} 0 & \text{if } a = b, \\ 1 & \text{if } a \neq b \text{ and } (a = \text{“}-\text{”} \text{ or } b = \text{“}-\text{”}), \\ \infty & \text{otherwise.} \end{cases}$$

Note that the edit distance of two sequences p_i and p_j in WMSA is the same as $\text{dist}(p_i, p_j)$ in BMP.

Solution for I implies solution for I'. Suppose we have a solution for I, i.e., an embedding ε. Note that $\varepsilon = \begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{pmatrix}$ is an alignment for $\mathcal{P}$ and the pairwise score of ε_i and ε_j is exactly $\text{conf}_\varepsilon(p_i, p_j)$. Therefore,

$$\begin{aligned} \text{SP}(\mathcal{P}', w) &= \frac{1}{2} \sum_{1 \le i,j \le n} w(i,j) \sum_{1 \le y \le |D|} \delta(\varepsilon_i[y], \varepsilon_j[y]) \\ &= \frac{1}{2} \sum_{1 \le i,j \le n} w(i,j) \text{conf}_\varepsilon(p_i, p_j) \\ &= \frac{1}{2} \sum_{p_i, p_j : \phi(p_j) \in \mathcal{N}(\phi(p_i))} \text{conf}_\varepsilon(p_i, p_j) \\ &= \text{BL}(\phi, \varepsilon)\,. \end{aligned}$$

Note that the second last equality is due to the definition of $w(i,j)$, which is based on the placement ϕ.

Solution for I' implies solution for I. On the other hand, suppose we have a solution for I', i.e., an alignment $\mathcal{P}' = \begin{pmatrix} p'_1 \\ \vdots \\ p'_n \end{pmatrix}$ for $\mathcal{P}$ and $|p'_i| = m'$, for some m'. In the alignment $\mathcal{P}'$, each column contains the same character or “ $-$ ” because

of the definition of the distance function $\delta(a, b)$. We denote the resulting matrix as $\varepsilon = (\varepsilon_1 \cdots \varepsilon_n)$. It can be seen that ε is an embedding for $\mathcal{P}$ and the hamming distance between ε_i and ε_j equals the pair-wise score of p'_i and p'_j. Then

$$\begin{aligned}
\mathrm{BL}(\phi, \varepsilon) &= \frac{1}{2} \sum_{p_i, p_j : \phi(p_j) \in \mathcal{N}(\phi(p_i))} \mathrm{conf}_\varepsilon(p_i, p_j) \\
&= \frac{1}{2} \sum_{p_i, p_j : \phi(p_j) \in \mathcal{N}(\phi(p_i))} \sum_{1 \le y \le |D|} \delta(p'_i[y], p'_j[y]) \\
&= \frac{1}{2} \sum_{1 \le i, j \le n} w(i, j) \sum_{1 \le y \le |D|} \delta(p'_i[y], p'_j[y]) \\
&= \mathrm{SP}(\mathcal{P}', w) .
\end{aligned}$$

Note that the second last equality holds for the same reason as above. Therefore, the lemma follows. □

With Lemma 16, the following corollary is a direct consequence of Lemma 15.

Corollary 17. *The P-BMP problem is $O(\log^2 n)$-approximable.*

5.2.2 Approximation for BMP

In this section, we study the BMP problem in which we are to find both the placement as well as the embedding to minimise the border length. We give an $O(\sqrt{n} \log^2 n)$-approximation, which makes use of the approximability result for P-BMP (Section 5.2.1). To make use of the result for P-BMP, we need a certain placement, the choice of which is guided by some travelling salesman path (TSP) on a particular graph (to be defined). Note that finding the minimum TSP is NP-hard, yet there is a polynomial time $O(1)$-approximation [21].

The algorithm PLACE&EMBED**.** The approximation algorithm PLACE&EMBED is shown in Algorithm 4. The graph G_c constructed in the algorithm is a weighted complete graph with vertices representing $\mathcal{P}$ and edge weight representing dist() between the two vertices. A travelling salesman path (TSP) is obtained from G_c, which

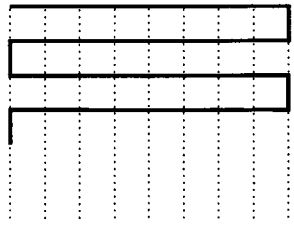

Figure 5.1: Row-by-row threading of a TSP (solid edges) on a grid. In the placement formed, solid edges connect neighbours in the placement that are also neighbours on the TSP and dotted edges otherwise.

we *"thread"* on the grid G in a row-by-row fashion to form a placement [39]: the TSP is placed from left to right on the first row, right to left on the second, and then alternate in the same way in the remaining rows (see Figure 5.1 for an example). In the placement formed, solid edges in Figure 5.1 connect neighbours in the placement that are also neighbours on the TSP while dotted edges connect neighbours in the placement that are not neighbours on the TSP. We then employ the approximation algorithm in Section 5.2.1. We denote the placement and embedding computed by PLACE&EMBED as $\tilde{\phi}$ and $\tilde{\varepsilon}$, respectively.

Algorithm 4 PLACE&EMBED: Approximation algorithm for BMP.

Input: Probe set $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ to be placed on a $\sqrt{n} \times \sqrt{n}$ array.

Output: A placement $\tilde{\phi}$ and an embedding $\tilde{\varepsilon}$ for $\mathcal{P}$.

Steps:

1: Construct the weighted complete graph G_c.
2: Find an approximate TSP $\tilde{Q}$ for G_c using algorithm in [21].
3: Thread $\tilde{Q}$ in a row-by-row fashion to obtain a placement $\tilde{\phi}$.
4: Run the approximation algorithm for P-BMP in Section 5.2.1 (i.e., by reducing the P-BMP instance to an WMSA instance) to obtain an embedding $\tilde{\varepsilon}$.

Analysis. The rest of this section is devoted to proving the following theorem.

Theorem 18. *Algorithm* PLACE&EMBED *is an* $O(\sqrt{n}\log^2 n)$*-approximation for the general BMP problem.*

To analyse the performance of PLACE&EMBED, we need some notations. Recall that we define for any sequences p, q, $\text{dist}(p, q) = 2(\ell - |LCS(p, q)|)$. We overload the notation dist() for any subgraph of G_c. For any subgraph H of G_c, we define the LCS distance of H, denoted by $\text{dist}(H)$, to be the sum of LCS distances of neighbouring probes in H, i.e., $\text{dist}(H) = \frac{1}{2}\sum_{p,\, q\, :\, q \in \mathcal{N}(p) \text{ in } H} \text{dist}(p, q)$.

As mentioned before in Section 5.1, $\text{dist}(p, q)$ is the minimum conflict between probes p and q. Yet the embeddings needed to achieve $\text{dist}(p, q)$ may not be compatible with each other in a particular placement. For example, consider the placement ϕ in Figure 1.5, $\text{dist}(\phi) = 8$ since $\text{dist}(p, q) = 2$ for every neighbouring pair p, q. Yet the minimum border length is 10 with CTAC as the deposition sequence, and embeddings $(--\text{AC}, -\text{TA}-, \text{CT}--, \text{C}-\text{A}-)$. We summarise this as follows.

Observation 19. *Given a placement ϕ, we have $dist(\phi) \leq BL(\phi, \varepsilon)$, for any embedding ε.*

Observation 19 implies that for the optimal placement ϕ^* and optimal embedding ε^*, $\text{dist}(\phi^*) \leq \text{BL}(\phi^*, \varepsilon^*)$. To approximate BMP, it suffices to bound the border length by $\text{dist}(\phi^*)$. On the other hand, we make the following observation about a graph H_1 and its subgraph H_2. The observation is true since any neighbours in H_1 are also neighbours in H_2 but not vice versa.

Observation 20. *Consider any graph H_1 and a subgraph H_2 of it. $dist(H_2) \leq dist(H_1)$.*

By Observation 20, we derive the next corollary about the optimal TSP Q^* and the optimal placement ϕ^*.

Corollary 21. *Suppose Q^* is the optimal TSP for G_c. Then, we have $dist(Q^*) \leq dist(\phi^*)$.*

Proof. ϕ^* can be viewed as threading a TSP Q in a row-by-row fashion. By Observation 20, we have $\text{dist}(Q) \leq \text{dist}(\phi^*)$. Since Q^* is the optimal TSP for G_c, we have $\text{dist}(Q^*) \leq \text{dist}(Q) \leq \text{dist}(\phi^*)$. □

It is known that TSP can be approximated by $3/2$ (Lemma 22). Therefore, $\text{dist}(\tilde{Q}) \leq 3\,\text{dist}(Q^*)/2$.

Lemma 22. [21] *The travelling salesman problem admits a $3/2$-approximation if the weight satisfies the triangle inequality.*

To complete the proof of Theorem 18, it remains to relate $\text{BL}(\tilde{\phi}, \tilde{\varepsilon})$ and $\text{dist}(\tilde{Q})$.

Lemma 23. *(i) $dist(\tilde{\phi}) \leq 2\sqrt{n}\, dist(\tilde{Q})$; and (ii) $BL(\tilde{\phi}, \tilde{\varepsilon}) \leq O(\log^2 n)\, dist(\tilde{\phi})$.*

Proof. (i) Suppose $\tilde{Q} = \{u_1, u_2, \ldots, u_n\}$. Note that the LCS distance dist() satisfies the triangular inequality, i.e., $\text{dist}(u_i, u_j) \leq \sum_{i \leq k < j} \text{dist}(u_k, u_{k+1})$. Neighbouring probes on $\tilde{Q}$ are also neighbours in $\tilde{\phi}$ but not vice versa. For any two probes u_i and u_j which are neighbours in $\tilde{\phi}$, we have $1 \leq |j - i| < 2\sqrt{n}$. When we sum up $\text{dist}(\tilde{\phi})$, $\text{dist}(u_k, u_{k+1})$, for any k, may be counted more than once, but no more than $2\sqrt{n}$ times. Therefore, $\text{dist}(\tilde{\phi}) \leq 2\sqrt{n}\,\text{dist}(\tilde{Q})$.

(ii) In Step 5.2.2 of algorithm PLACE&EMBED, we reduce the P-BMP instance with $\tilde{\phi}$ as the given placement to an WMSA instance. Lemma 16 asserts that the border length of the resulting embedding obtained is the same as the weighted SP score of the alignment. Furthermore, we have seen in Section 5.1 that approximation for WMSA can be found by the approximation for MRCT and the resulting routing tree has a routing cost, and thus, the weighted SP score, at most $O(\log^2 n)$ times the total weighted edit distance in WMSA. In the proof of Lemma 16, we note that the weighted edit distance of two sequences is the same as dist() of the two sequences. As a result, $\text{BL}(\tilde{\phi}, \tilde{\varepsilon}) \leq O(\log^2 n)\,\text{dist}(\tilde{\phi})$. □

We now complete the proof of Theorem 18.

Proof of Theorem 18. By Lemmas 23, 22, and Corollary 21, we have

$$\begin{aligned} \text{BL}(\tilde{\phi}, \tilde{\varepsilon}) &\leq O(\sqrt{n}\log^2 n)\,\text{dist}(\tilde{Q}) \\ &\leq O(\sqrt{n}\log^2 n)\,\text{dist}(Q^*) \\ &\leq O(\sqrt{n}\log^2 n)\,\text{dist}(\phi^*)\,. \end{aligned}$$

Furthermore, Observation 19 holds for all placement, and hence for ϕ^*, in other words, $\text{dist}(\phi^*) \leq \text{BL}(\phi^*, \varepsilon^*)$. Therefore, $\text{BL}(\tilde{\phi}, \tilde{\varepsilon}) \leq O(\sqrt{n}\log^2 n)\,\text{BL}(\phi^*, \varepsilon^*)$. □

5.2.3 One dimensional array

In this section, we study the special case on an 1D array. Intuitively, the problem is easier than the 2D case. We show that P-BMP on an 1D array can be solved optimally in polynomial time while BMP on an 1D array admits an O(1)-approximation.

P-BMP on 1D array. The algorithm EMBED1D shown in Algorithm 6 makes use of a procedure called EXTEND (see Algorithm 5). EXTEND takes two sequences p and q, and a supersequence S of p as input and returns a supersequence of S and q. Let $c = |LCS(p,q)|$, $x_1, x_2, \ldots, x_c$ be the indices of S corresponding to p that belongs to $LCS(p,q)$, and $y_1, y_2, \ldots, y_c$ be the indices of q that belongs to $LCS(p,q)$. EXTEND then extends S by inserting characters in q but not in $LCS(p,q)$: characters between $q[y_{k-1}]$ and $q[y_k]$ are inserted right before $S[x_k]$ and characters beyond $q[y_c]$ are appended to the end of S. Furthermore, EXTEND keeps track of the indices of the new S that correspond to q. See Figure 5.2 for an illustration.

Algorithm 5 EXTEND: Procedure for computing supersequence.

Input: Two sequences p, q, and a supersequence S of p.
Output: A supersequence of S and q.
Steps:

1: Find $LCS(p,q)$ and set $c = |LCS(p,q)|$.
2: Set $x_1, x_2, \ldots, x_c$ be the indices of S corresponding to p that belongs to $LCS(p,q)$ and $y_1, y_2, \ldots, y_c$ be the indices of q that belongs to $LCS(p,q)$.
3: Extend S by inserting characters in q but not in $LCS(p,q)$: characters between $q[y_{k-1}]$ and $q[y_k]$ are inserted right before $S[x_k]$ and characters beyond $q[y_c]$ are appended to the end of S.
4: Keep track of the indices of the new S that correspond to q.

Example: Let sequences $p = \text{CAT}, q = \text{ACT}$ and the supersequence S of p where

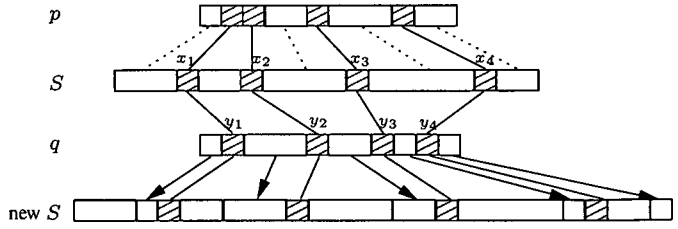

Figure 5.2: An illustration of the procedure EXTEND. The shaded squares refer to characters in $LCS(p, q)$. Characters in q but not in $LCS(p, q)$ are inserted into S so that the order preserves as in q (indicated by the arrows).

S = CACTG be the input. EXTEND finds the LCS between p and q which is AT. Then, $x_1 = 2, x_2 = 4$ are the indices of S corresponding to p that belongs to $LCS(p, q)$ and $y_1 = 1, y_2 = 3$ be the indices of q that belongs to $LCS(p, q)$. Then, the character C in q needs to be inserted into S before $S[x_2]$ which is $S[4]$. As a result, the new S = CACCTG and indices $2, 4, 5$ of the new S that corresponds to q are kept by EXTEND.

Algorithm 6 EMBED1D: Optimal embedding for P-BMP on 1D array.

Input: Probe set $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$, placed on a 1D array in that order.
Output: An embedding ε with minimum border length.
Steps:

1: Set $D = p_1$.
2: For $i > 1$, call the procedure EXTEND with p_{i-1}, p_i and D as the input to obtain a new D.
3: For each p_i, set ε_i such that $\varepsilon_i[y] = D[y]$ if $D[y]$ corresponds to a character in p_i kept track by EXTEND, and $\varepsilon_i[y] =$ " $-$ " otherwise.

The algorithm EMBED1D takes a probe set $\mathcal{P}$ with n probes placed on a 1D array in that order such that p_i and p_{i+1} are neighbours, for $1 \leq i < n$, as input. EMBED1D finds a common supersequence D for $\mathcal{P}$ iteratively, and an embedding can then be obtained from D. Initially, D is simply p_1. Then, we call the procedure EXTEND with p_{i-1}, p_i and D as the input to obtain a new D until EMBED1D finds a common supersequence D for all probes in $\mathcal{P}$. For each p_i, we define ε_i such that

$\varepsilon[y] = D[y]$ if $D[y]$ corresponds to a character in p_i kept track by EXTEND, and $\varepsilon[y] =$ " $-$ " otherwise.

Example: Suppose we have a probe set $\mathcal{P} = \{$AC, TA, CG, TC$\}$ that are placed on the 1D array in that order. Then D starts with AC, changes iteratively to TAC, TACCG, and the final is TACTCG. Note that the common supersequence obtained at the end is not necessarily the shortest one. The corresponding embeddings are ($-$AC$---$), (TA$----$), ($----$CG), and ($---$TC$-$).

We state the performance of EMBED1D.

Theorem 24. EMBED1D *finds an optimal embedding for the P-BMP problem on 1D array in polynomial time.*

Proof. We first observe that D constructed in each iteration by EXTEND is a common supersequence of $p_1, \ldots, p_i$. This is clear from the way EXTEND finds $LCS(p_{i-1}, p_i)$ and then inserts into D those characters in p_i that are not in the LCS while preserving the order of appearance as in p_i. It also implies that the number of nucleotides shared by p_{i-1} and p_i is maintained as $|LCS(p_{i-1}, p_i)|$, which is the maximum possible. Note that this property does not change by later steps. Hence, the border length of the final embedding is the minimum.

As for time complexity, it is known that longest common subsequence of two sequences can be found in $O(\ell^2)$ time using straightforward dynamic programming [41]. EMBED1D finds pair-wise common supersequence for $n-1$ time and inserting characters in each step takes $O(n\ell)$ time. As a result, the time complexity of EMBED1D is $O(n\ell \max\{n, \ell\})$. The time complexity can further be improved by using the $O(\ell \log \ell)$ time dynamic programming [67, 91]. □

BMP on 1D array. When the placement is not given (i.e., BMP), we can extend the algorithm EMBED1D to find both a placement and embedding for the probe set $\mathcal{P}$. The approximation algorithm devised is called PLACE&EMBED1D (see Algorithm 7). Similar to the case on 2D array (see Section 5.2.2), we first find a

placement by (i) constructing the weighted complete graph G_c, (ii) finding an approximate TSP $\bar{Q}$ on G_c, and (iii) threading $\bar{Q}$ in the TSP order on the 1D array to obtain a placement $\bar{\phi}$. Given this placement $\bar{\phi}$, we find an optimal embedding $\bar{\varepsilon}$ by EMBED1D. We then show in Theorem 25 that $\mathrm{BL}(\bar{\phi}, \bar{\varepsilon})$ is at most $3/2$ times the optimal border length.

Algorithm 7 PLACE&EMBED1D: Approximation algorithm for BMP on 1D array.

Input: Probe set $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ to be placed on a 1D array.

Output: A placement $\bar{\phi}$ and an embedding $\bar{\varepsilon}$ for $\mathcal{P}$.

Steps:

1: Construct the weighted complete graph G_c.
2: Find an approximate TSP $\bar{Q}$ for G_c using algorithm in [21].
3: Thread $\bar{Q}$ in the TSP order on the 1D array to obtain a placement $\bar{\phi}$.
4: Run EMBED1D to obtain an optimal embedding $\bar{\varepsilon}$.

Theorem 25. *There is a polynomial time algorithm for BMP on 1D array with approximation ratio* $3/2$.

Proof. Note that on 1D array, neighbouring probes on $\bar{Q}$ are also neighbours in $\bar{\phi}$ and visa versa. Therefore, $\mathrm{dist}(\bar{\phi}) = \mathrm{dist}(\bar{Q})$. Since EMBED1D finds the optimal embedding $\bar{\varepsilon}$ with $\bar{\phi}$ as the given placement, we have $\mathrm{BL}(\bar{\phi}, \bar{\varepsilon}) = \mathrm{dist}(\bar{\phi}) = \mathrm{dist}(\bar{Q})$. Combining with Lemma 22, Corollary 21 and Observation 19, we have

$$\begin{aligned} \mathrm{BL}(\bar{\phi}, \bar{\varepsilon}) &= \mathrm{dist}(\bar{Q}) \\ &\leq 3\,\mathrm{dist}(Q^*)/2 \\ &\leq 3\,\mathrm{dist}(\phi^*)/2 \\ &\leq 3\,\mathrm{BL}(\phi^*, \varepsilon^*)/2\,. \end{aligned}$$

As a result, PLACE&EMBED1D gives a $3/2$-approximation for BMP on 1D array. □

5.3 The maximum agreement problem (MAP)

In this section, we study the counter part of BMP, which is called maximum agreement problem (MAP) (recall definition in Section 5.1). In contrast to BMP, we show that MAP admits constant approximations, regardless of whether the placement is given in advance or not.

5.3.1 Approximation for P-MAP

We first study the special case of the MAP problem when the placement of probes is given in advance, i.e., the placement is part of input. The P-MAP problem (MAP problem with given placement) is to find an embedding to maximise the agreement. We are going to show that P-MAP admits an $O(1)$-approximation algorithm. The algorithm devised is called AEMBED (see Algorithm 8).

Algorithm AEMBED**.** The algorithm AEMBED (EMBED for Agreement) makes use of procedure EXTEND in Section 5.2.3. The order of probes to be considered is determined by a certain tree τ with the bottom rightmost probe in G being the root. To construct τ, for each probe p, we assign a parent to the probe, denoted by $parent(p)$. We denote by $r(p)$ and $b(p)$ the right and bottom neighbours of probe p, respectively. The probes in the rightmost column and bottommost column have $r(p) =$ NULL and $b(p) =$ NULL, respectively. We set $parent(p)$ to $r(p)$ or $b(p)$ depending on whether $|LCS(p, r(p))|$ or $|LCS(p, b(p))|$ is larger.

After tree τ is constructed, AEMBED finds a common supersequence D for $\mathcal{P}$ by traversing τ in a pre-order fashion recursively. Initially, D is set to be the bottom rightmost probe in the grid G. When a certain depth i is reached, AEMBED calls the procedure EXTEND with $parent(p)$, p and D as input. Finally, for each p_i, we define $\hat{\varepsilon}_i$ such that $\hat{\varepsilon}_i[y] = D[y]$ if $D[y]$ corresponds to a character in p_i kept track by EXTEND, and $\hat{\varepsilon}_i[y] =$ " $-$ " otherwise.

Figure 5.3 shows an example. Suppose we have a set of probes $\mathcal{P} =$ {ACT, CTG, CAT, GAC, GCC, AAT, TAC, GCT, CTT} placed on a 3×3 grid G (see Figure 5.3

Algorithm 8 AEMBED: Approximate algorithm for P-MAP.

Input: Probe set $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ placed on a $\sqrt{n} \times \sqrt{n}$ array according to a given placement ϕ.

Output: An embedding $\hat{\varepsilon}$ for $\mathcal{P}$.

Steps:

1: Construct a tree τ by assigning parent to each probe p: if $|LCS(p, r(p))| \geq |LCS(p, b(p))|$ set $parent(p) = r(p)$ else set $parent(p) = b(p)$.
2: Set D to be the bottom rightmost probe in the grid G.
3: Traverse τ in a pre-order fashion: for each probe p traversed, call the procedure EXTEND with $parent(p)$, p and D as input.
4: For each p_i, set $\hat{\varepsilon}_i$ such that $\hat{\varepsilon}_i[y] = D[y]$ if $D[y]$ corresponds to a character in p_i kept track by EXTEND, and $\hat{\varepsilon}_i[y] =$ " $-$ " otherwise.

(a)). The values on the edges represent the length of the longest common subsequence between the two neighbouring probes. For a probe p, we set $parent(p)$ to $r(p)$ if $|LCS(p, r(p))| \geq |LCS(p, b(p))|$, otherwise set $parent(p) = b(p)$. For example, the parent of probe $p =$ CTG is set to be $r(p) =$ CAT since $|LCS(p, r(p))| > |LCS(p, b(p))|$. As for probe $p =$ GCC, we set $parent(p)$ to $b(p) =$ GCT because $|LCS(p, b(p))| > |LCS(p, r(p))|$. An arrow from p to q in Figure 5.3 (a) means that $parent(p) = q$. Figure 5.3 (b) shows the tree τ constructed by AEMBED with root CTT. The deposition sequence D starts with the root CTT. Since AEMBED traverses tree τ in a pre-order fashion, the first traversed probe is AAT. EXTEND computes the supersequence for AAT and CTT which is CAATT. Similarly, EXTEND computes the supersequence with the next traversed probe CAT, its parent AAT and the previous D =CAATT as input, returning new $D =$ CCAATT. In such a way, the deposition sequence D changes iteratively and finally $D =$ GATACACAATTGC as shown in Figure 5.3 (c).

In this example, the agreement is 30, while the optimal agreement is no more than 38 which is twice of the sum of the LCS distances over the edges. The analysis of the performance of AEMBED is as follows.

Analysis. To analyse the performance of AEMBED, we first observe that in

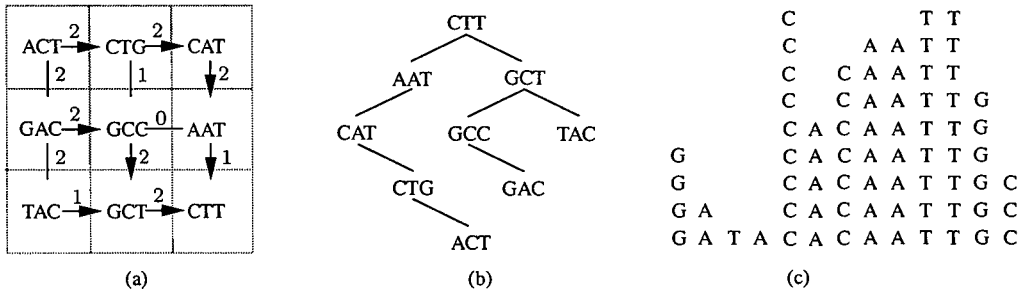

Figure 5.3: (a) A set of probes placed on a 3×3 grid G. The edge labels are length of LCS and the arrows point to the parent. (b) The tree τ constructed by AEMBED with root CTT. (c) Iterative changes of the deposition sequence D (character are drawn to align with the final D.

the final embedding $\hat{\varepsilon}$, the number of nucleotides shared by a probe and its parent equals to the length of their LCS by the property of EXTEND. We then bound the performance of AEMBED as follows.

Theorem 26. AEMBED *is a polynomial-time 2-approximation algorithm for the P-MAP problem.*

Proof. For the given placement ϕ and the optimal embedding ε^*, the optimal agreement is:

$$\mathrm{A}(\phi, \varepsilon^*) = \sum_{p \in \mathcal{P}} (\mathrm{share}_{\varepsilon^*}(p, r(p)) + \mathrm{share}_{\varepsilon^*}(p, b(p))) .$$

We assume $\mathrm{share}_{\varepsilon^*}(p, q) = 0$ if $q = \mathrm{NULL}$. As mentioned in Section 5.1, for any embedding, the share between the embeddings of probes p, q is at most $2|LCS(p, q)|$. Thus, $2|LCS(p, r(p))| \geq \mathrm{share}_{\varepsilon^*}(p, r(p))$ and $2|LCS(p, b(p))| \geq \mathrm{share}_{\varepsilon^*}(p, b(p))$. Note that

$$\begin{aligned} \mathrm{share}_{\hat{\varepsilon}}(p, parent(p)) &= 2 \max\{|LCS(p, r(p))|, |LCS(p, b(p))|\} \\ &\geq \frac{1}{2}(\mathrm{share}_{\varepsilon^*}(p, r(p)) + \mathrm{share}_{\varepsilon^*}(p, b(p))) . \end{aligned}$$

Therefore,

$$\begin{aligned} \mathrm{A}(\phi, \hat{\varepsilon}) &= \sum_{p \in \mathcal{P}} \mathrm{share}_{\hat{\varepsilon}}(p, parent(p)) \\ &\geq \frac{1}{2}\mathrm{A}(\phi, \varepsilon^*) . \end{aligned}$$

Finally, AEMBED runs in polynomial time as the bottleneck is finding longest common subsequences between two sequences. As discussed in Section 5.2.3, the longest common subsequences of two sequences can be found in time $O(\ell \log \ell)$. □

5.3.2 Approximation for MAP

In this section, we study the general MAP problem to find both the placement and the embedding to maximise the agreement. We prove that the proposed algorithm APLACE&EMBED as shown in Algorithm 9 has an asymptotic approximation ratio of 4.

Algorithm 9 APLACE&EMBED: Approximation algorithm for MAP.

Input: Probe set $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ to be placed on a $\sqrt{n} \times \sqrt{n}$ array.
Output: A placement $\check{\phi}$ and an embedding $\check{\varepsilon}$ for $\mathcal{P}$.
Steps:

1: Partition $\mathcal{P}$ into four disjoint groups G_A, G_C, G_G and G_T: a probe belongs to G_A if the number of A in the probe is the maximum over the number of other characters (similarly for G_C, G_G and G_T).
2: Thread the probes in group G_A on the array in a row-by-row fashion, followed by threading of probes in G_C, G_G, and G_T to form the placement $\check{\phi}$.
3: For probes in G_A, align them such that the maximum number of A are aligned while different characters are not aligned. This forms a partial embedding $\check{\varepsilon}_a$ with deposition sequence D_a. Similarly, find $\check{\varepsilon}_c$, $\check{\varepsilon}_g$, $\check{\varepsilon}_t$ and D_c, D_g, D_t.
4: Combine D_a, D_c, D_g, and D_t to form D (append one after the other).
5: Extend the embeddings $\check{\varepsilon}_a$, $\check{\varepsilon}_c$, $\check{\varepsilon}_g$, $\check{\varepsilon}_t$ according to D by inserting " $-$ " in the columns corresponding to other groups. The union of the extended embeddings is the resulting embedding $\check{\varepsilon}$.

Suppose that we are given a probe set $\mathcal{P}$ with n probes. Since there are only four

possibilities for each character in a probe, there must be a particular character such that the number of the character is at least $\ell/4$ in the probe. We make use of this property to divide $\mathcal{P}$ into four groups G_A, G_C, G_G and G_T, such that the number of characters A, C, G, T of each probe in corresponding groups G_A, G_C, G_G, G_T is at least $\ell/4$. APLACE&EMBED first threads the probes in group G_A on the array in a row-by-row fashion, followed by threading of probes in G_C, G_G, and G_T to form the placement $\check{\phi}$. Then, APLACE&EMBED finds an alignment of probes in group G_A such that only characters A are aligned. This forms a partial embedding $\check{\varepsilon}_a$ with deposition sequence D_a. Similarly, algorithm APLACE&EMBED finds $\check{\varepsilon}_c$, $\check{\varepsilon}_g$, $\check{\varepsilon}_t$ and D_c, D_g, D_t. The deposition sequences D_a D_c, D_g, D_t are concatenated together by appending one after the other to form the final deposition sequence D. Finally, we extend the embeddings $\check{\varepsilon}_a$, $\check{\varepsilon}_c$, $\check{\varepsilon}_g$, $\check{\varepsilon}_t$ according to D by inserting " $-$ " in the columns corresponding to other groups. The union of the extended embeddings is the resulting embedding $\check{\varepsilon}$.

Figure 5.4 shows an example. Suppose we have a set of probes $\mathcal{P} = \{$ACAT, CTCT, GCAG, ATGT, AATG, GCCA, GGGA, CTTT, GACA$\}$ to be placed on a 3×3 array (see Figure 5.4 (a)). The probe set $\mathcal{P}$ is partitioned into groups $G_A = \{$ACAT, AATG, GACA$\}$, $G_C = \{$CTCT, GCCA$\}$, $G_G = \{$GCAG, GGGA$\}$ and $G_T = \{$CTTT, ATGT$\}$ (see Figure 5.4 (b)). Then, we thread the probes in group G_A on the array in a row-by-row fashion, followed by threading of probes in G_C, G_G, and G_T to form the placement $\check{\phi}$ as shown in Figure 5.4 (c). The embedding of each group is demonstrated in Figure 5.4 (d) and the corresponding deposition sequences are $D_A =$ GACCATTG, $D_C =$ GCTCTA, $D_G =$ GCAGGA and $D_T =$ CATGTT. The final deposition sequence D and embedding are displayed in Figure 5.4 (e).

Theorem 27. *The asymptotic approximation ratio of* APLACE&EMBED *is* 4.

Proof. Consider the optimal placement ϕ^* and embedding ε^*. It is obvious that for every pair of neighbouring probes p, q, $\text{share}_{\varepsilon}(p, q) \leq 2\ell$. There are a total of $2(n - \sqrt{n})$ pairs of neighbours on the grid in total. Therefore, the optimal agreement

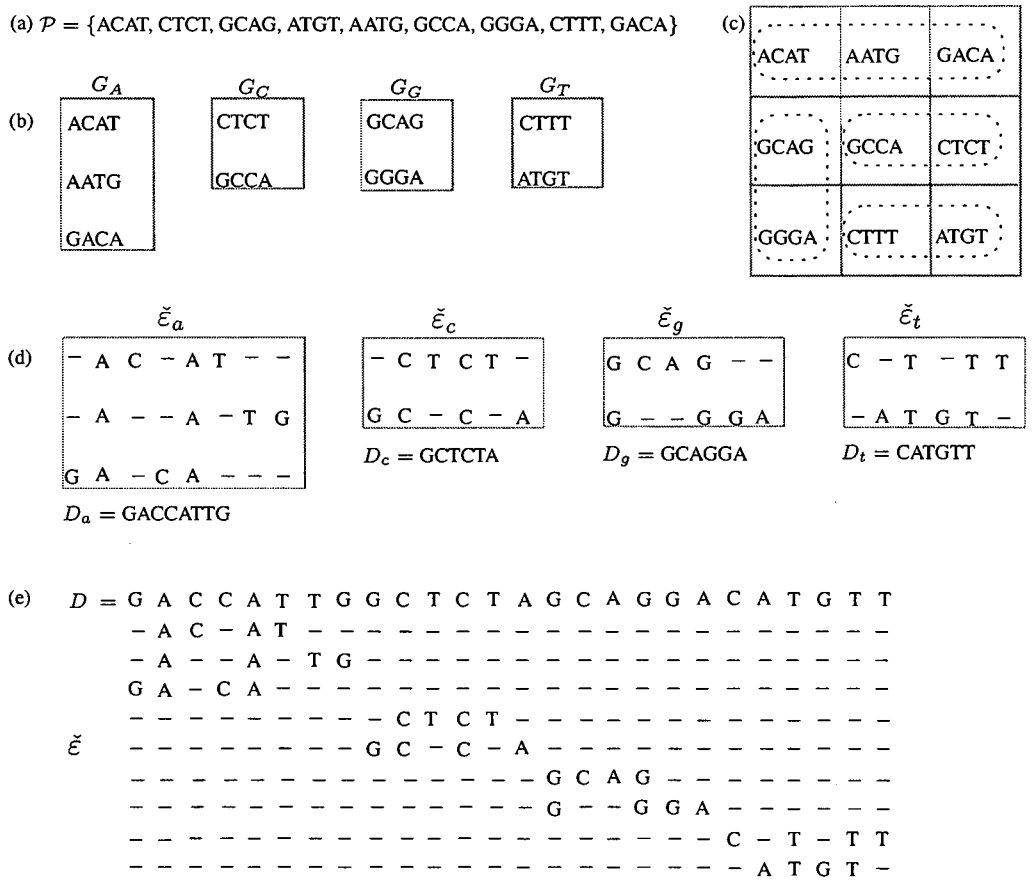

Figure 5.4: An illustration to APLACE&EMBED.

$\mathrm{A}(\phi^*, \varepsilon^*) \leq 4\ell(n - \sqrt{n})$.

On the other hand, consider $\check{\phi}$ and $\check{\varepsilon}$ returned by APLACE&EMBED. According to the way we partition the probes into group, for any two probes p, q in a group, the number of nucleotides that can be shared is at least $\ell/4$. Hence, $\mathrm{share}_{\check{\varepsilon}}(p, q) \geq 2(\ell/4) = \ell/2$. As we have seen above, there are altogether $2(n - \sqrt{n})$ pairs of neighbours in the grid. We may not share any nucleotide when the pair belongs to different groups. According to the way we thread the groups, there are at most $3\sqrt{n} + 3$ such pairs ($\sqrt{n}$ pairs of vertical neighbours between consecutive groups

and 3 pairs of neighbours that are the last one in a group and the first one in the next group). As a result, we have at least $2n - 5\sqrt{n} - 3$ pairs each with $\text{share}_{\check{\varepsilon}}()$ at least $\ell/2$. Therefore, $\text{A}(\check{\phi}, \check{\varepsilon}) \geq \ell(n - 2.5\sqrt{n} - 1.5)$. Then $\text{A}(\check{\phi}, \check{\varepsilon})/\text{A}(\phi^*, \varepsilon^*)$ tends to 4 as $\text{A}(\phi^*, \varepsilon^*)$ tends to infinity. So, the asymptotic approximation ratio of APLACE&EMBED is 4.

As for complexity, the bottleneck is to find an alignment in each group such that the maximum number of character in that group is aligned, which can be done in polynomial time. Therefore, APLACE&EMBED also takes polynomial time. □

5.4 Discussion

In this chapter, we have shown that the border minimisation problem (BMP) is $O(\sqrt{n}\log^2 n)$-approximable. We further show that a variant of BMP problem, in which a placement is given in advance (P-BMP), is $O(\log^2 n)$-approximable. On the other hand, we show that BMP can be defined as the maximum agreement problem (MAP) and we are able to devise $O(1)$-approximation algorithms for MAP regardless of whether the placement is given in advance or not.

The BMP problem is believed to be NP-hard with no known NP-hardness proof. An open question is to derive an NP-hardness proof. Another interesting open question is to improve the approximation ratio and/or derive inapproximability result. As mentioned before, there is an exponential time algorithm to compute the optimal BMP solution. Improving the exponential time algorithm could be useful in practice and is of theoretical interest.

Chapter 6

Conclusion

In this thesis, we have studied three problems arising from DNA microarray design and presented new combinatorial algorithms for the considered problems. In this final chapter, we will summarise the work and result described in each chapter, and give some future directions.

In Chapter 3, We have proposed a new approach to select a small set of probes by using randomisation and demonstrated that such a small set of probes is sufficient to distinguish each gene from all the other genes in the dataset. We believe that the proposed approach should be useful in the design of a robust collection of multiple probes. Another possible direction of study is further investigations on sensitivity that may lead to more accurate prediction of DNA secondary structure and free energy models.

In Chapter 4, we investigate a variant of the string barcoding problem in which the probes have to be chosen from a particular set of probes of cardinality n, such that an appropriate combination of probes with minimum cardinality is used to distinguish all genes in a dataset $\mathcal{S}$. We present almost optimal $O(n|\mathcal{S}|\log^3 n)$-time approximation algorithm for the considered problem. An interesting direction would be the design of fault-tolerant barcodes in which every pair of strings are separated by two (or more) probes available in the pool of precomputed probes.

In Chapter 5, we show that BMP admits an $O(\sqrt{n}\log^2 n)$-approximation, where

n is the number of probes to be synthesised. In the case where the placement is given in advance, we show that the problem is $O(\log^2 n)$-approximable. We also study a related problem called agreement maximisation problem (MAP). In contrast to BMP, we show that MAP admits a constant approximation even when placement is not given in advance. The border minimisation problem is believed to be NP-hard with no known NP-hardness proof. An open question is to derive an NP-hardness proof. Another interesting open question is to improve the approximation ratio and/or derive inapproximability result. Improving the exponential time algorithm to compute the optimal BMP solution is also challenging, which could be useful in practice and is of theoretical interest.

Bibliography

[1] G. M. Adelson-Velskii and Y. M. Landis. *An algorithm for the organization of information.* Proceedings of the USSR Academy of Sciences 146: 263–266 (Russian). English translation by M. J. Ricci in Soviet Math. Doklady, 3:1259–1263, 1962.

[2] P. Agarwal and D. J. States. Comparative accuracy of methods for protein sequence similarity search. *Bioinformatics*, 14:40–47, 1998.

[3] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data structures and algorithms.* Addison-Wesley Publishing Company, 1982.

[4] N. Alon, C. J. Colbourn, A. C. H. Ling, and M. Tompa. Equireplicate balanced binary codes for oligo arrays. *SIAM Journal on Discrete Mathematics*, 14(4):481–497, 2001.

[5] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–402, 1997.

[6] V. Bafna, E. L. Lawler, and P. A. Pevzner. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *Journal of Molecular Evolution*, 25(4):351–360, 1997.

[7] T. L. Bailey and M. Gribskov. Combining evidence using p-values:application to sequence homology searches. *Bioinformatics*, 14:48–54, 1998.

[8] W. Bains and G. Smith. A novel method for nucleic acid sequence determination. *Journal of Theoretical Biology*, 135:303–307, 1988.

[9] Y. Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of the Thirty-Seventh Symposium on Foundations of Computer Science*, pages 184–193, 1996.

[10] P. Berman, B. DasGupta, and M. Y. Kao. Tight approximability results for test set problems in bioinformatics. *Journal of Computer and System Sciences*, 71(2):145–162, 2005.

[11] P. Bonizzoni and G. D. Vedova. The complexity of multiple sequence alignment with SP-score that is a metric. *Theoretical Computer Science*, 259(1–2):63–79, 2001.

[12] J. Borneman, M. Chrobak, G. D. Vedova, A. Figueroa, and T. Jiang. Probe selection algorithms with applications in the analysis of microbial communities. *Bioinformatics*, 17:S39–S48, 2001.

[13] R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Communications of the Association for Computing Machinery*, 20(10):762–772, 1977.

[14] Z. Bozdech, J. Zhu, M. P. Joachimiak, F. E. Cohen, B. Pulliam, and J. L. DeRisi. Expression profiling of the schizont and trophozoite stages of *plasmodium falciparum* with a long-oligonucleotide microarray. *Genome Biology*, 4:R9, 2003.

[15] P. J. Cameron. *Combinatorics : Topics, Techniques, Algorithms.* Cambridge University Press, 1994.

[16] C. R. Cantor and P. R. Schimmel. *Biophysical Chemistry Part III: The Behavior of Biological Macromolecules.* W. H. Freeman, San Francisco, CA, 1980.

[17] S. A. Carvalho Jr. and S. Rahmann. Improving the layout of oligonucleotide. microarrays: Pivot partitioning. In *Proceedings of the Sixth Workshop on Algorithms in Bioinformatics (WABI)*, pages 321–332, 2006.

[18] S. A. Carvalho Jr. and S. Rahmann. Microarray layout as quadratic assignment problem. In *Proceedings of the German Conference on Bioinformatics (GCB)*, pages 11–20, 2006.

[19] S. A. Carvalho Jr. and S. Rahmann. Improving the design of genechip arrays by combining placement and embedding. In *Proceedings of the Sixth Computational Systems Bioinformatics (CSB)*, pages 54–63, 2007.

[20] M. Charikar, C. Chekuri, A. Goel, and S. Guha. Rounding via trees: deterministic approximation algorithms for group Steiner trees and k-median. In *Proceedings of the Thirtieth Symposium on Theory of Computing*, pages 114–123, 1998.

[21] N. Christofides. *Worst-case analysis of a new heuristic for the travelling salesman problem.* Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA. *(ND33)*, 1976.

[22] J. M. Claverie. Effective large-scale sequence similarity searches. *Methods in Enzymology*, 266:212–227, 1996.

[23] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* The MIT Press, 2001.

[24] B. DasGupta, K. M. Konwar, I. I. Mandoiu, and A. A. Shvartsman. Dnabar: distinguisher selection for dna barcoding. *Bioinformatics*, 21(16):3424–3426, 2005.

[25] B. DasGupta, K. M. Konwar, I. I. Mandoiu, and A. A. Shvartsman. Highly scalable algorithms for robust string barcoding. *International Journal of Bioinformatics Research and Applications.*, 1(2):145–161, 2005.

[26] Y. Ding, C. Y. Chan, and C. E. Lawrence. Sfold web server for statistical folding and rational design of nucleic acids. *Nucleic Acids Research*, 32:W135–W141, 2004.

[27] R. Drmanac, I. Labat, I. Brukner, and R. Crkvenjakov. Sequencing of megabase plus DNA by hybridization: theory of the method. *Genomics*, 4:114–128, 1989.

[28] D. F. Feng and R. F. Doolittle. Approximation algorithms for multiple sequence alignment. *Theoretical Computer Science*, 182(1):233–244, 1987.

[29] S. Fodor, J. L. Read, M. C. Pirrung, L. Stryer, A. T. Lu, and D. Solas. Light-directed, spatially addressable parallel chemical synthesis. *Science*, 251(4995):767–773, 1991.

[30] C. B. Fraser and R. W. Irving. Approximation algorithms for the shortest common supersequence. *Nordic Journal of Computing*, 2(3):303–325, 1995.

[31] A. G. Frutos, Q. Liu, A. J. Thiel, A. M. W. Sanner, A. E. Condon, L. M. Smith, and R. M. Corn. Demonstration of a word design strategy for DNA computing on surfaces. *Nucleic Acids Research*, 25(23):4748–4757, 1997.

[32] D. Gerhold, T. Rushmore, and C. T. Caskey. DNA chips: promising toys have become powerful tools. *Trends in Biochemical Sciences*, 24(5):168–173, 1999.

[33] L. Gąsieniec, C. Y. Li, and M. Zhang. *Faster Algorithm for the Set Variant of the String Barcoding Problem.* submitted, 2008.

[34] L. Gąsieniec, C.Y. Li, P. Sant, and P.W.H. Wong. Efficient probe selection in microarray design. In *Proceedings of the IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (IEEE CIBCB)*, pages 247–254, 2006.

[35] L. Gąsieniec, C.Y. Li, P. Sant, and P.W.H. Wong. Randomized probe selection algorithm for microarray design. *Journal of Theoretical Biology*, 248(3):512–521, 2007.

[36] D. Gusfield. Efficient methods for multiple sequence alignment with guaranteed error bounds. *Bulletin of Mathematical Biology*, 55(1):141–154, 1993.

[37] D. Gusfield. *Algorithms on String, Trees, and Sequences.* Cambridge University Press, New York, 1999.

[38] R. W. Hamming. Error-detecting and error-correcting codes. *Journal of the ACM (JACM)*, 46(3):395–415, 1999.

[39] S. Hannenhalli, E. Hubell, R. Lipshutz, and P. A. Pevzner. Combinatorial algorithms for design of dna arrays. *Advances in Biochemical Engineering/Biotechnology*, 77:1–19, 2002.

[40] D. Harel. *Algorithmics: the Spirit of Computing.* Addison-Wesley, Reading MA, 1987.

[41] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18(6):341–343, 1975.

[42] D.S. Hochbaum. *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, Boston, MA, 1997.

[43] I. L. Hofacker. Vienna RNA secondary structure server. *Nucleic Acids Research*, 31(13):3429–3431, 2003.

[44] E. Hubbell and P. A. Pevzner. Fidelity probes for dna arrays. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 113–117, 1999.

[45] D. S. Johnson, J. K. Lenstra, and A. H. G. Rinnooy Kan. The complexity of the network design problem. *Networks*, 8(4):279–285, 1978.

[46] L. Kaderali and A. Schliep. Selecting signature oligonucleotides to identify organisms using DNA arrays. *Bioinformatics*, 18:1340–1349, 2002.

[47] A. B. Kahng, I. I. Mandoiu, P. A. Pevzner, S. Reda, and A. Zelikovsky. Engineering a scalable placement heuristic for DNA probe arrays. In *Proceedings of the Seventh Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, pages 148–156, 2003.

[48] A. B. Kahng, I. I. Mandoiu, P. A. Pevzner, S. Reda, and A. Zelikovsky. Scalable heuristics for design of DNA probe arrays. *Journal of Computational Biology*, 11(2/3):429–447, 2004.

[49] A. B. Kahng, I. I. Mandoiu, P. A. Pevzner, S. Reda, and A. A. Zelikovsky. Border length minimization in DNA array design. In *Proceedings of the Second Workshop on Algorithms in Bioinformatics (WABI)*, pages 435–448, 2002.

[50] A. B. Kahng, I. I. Mandoiu, S. Reda, X. Xu, and A. Zelikovsky. Computer-aided optimization of DNA array design and manufacturing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(2):305–320, 2006.

[51] A. B. Kahng, I. I. Mandoiu, S. Reda, X. Xu, and A. A. Zelikovsky. Evaluation of placement techniques for DNA probe array layout. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 262–269, 2003.

[52] R. M. Karp, R. E. Miller, and A. L. Rosenberg. Rapid identification of repeated patterns in strings, trees and arrays. In *Proceedings of the Fourth Symposium on Theory of Computing*, pages 125–136, 1972.

[53] S. Kasif, Z. Weng, A. Detri, R. Beigel, and C. DeLisi. A computational framework for optimal masking in the synthesis of oligonucleotide microarrays. *Nucleic Acids Research*, 30(20):e106, 2002.

[54] G. W. Klau, S. Rahmann, A. Schliep, M. Vingron, and K. Reinert. Optimal robust non-unique probe selection using Integer Linear Programming. *Bioinformatics*, 20:i186–i193, 2004.

[55] D. E. Knuth, J. H. Morris Jr., and V. R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[56] G. Lancia and R. Rizzi1. The approximability of the string barcoding problem. *Algorithms for Molecular Biology*, 1(12):doi: 10.1186/1748–7188–1–12, 2006.

[57] Z. J. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23:337–343, 1977.

[58] C. Y. Li, P. W. H. Wong, Q. Xin, and F. C. C. Yung. *Approximating Border Length for DNA Microarray Synthesis.* submitted, 2008.

[59] F. Li and G. Stormo. Selection of optimal DNA oligos for gene expression arrays. *Bioinformatics*, 17(11):1067–1076, 2001.

[60] M. Li, H. J. Lee, A. E. Condon, and R. M. Corn. DNA word design strategy for creating sets of non-interacting sets of oligonucleotides for DNA microarrays. *Langmuir*, 18(3):805–812, 2002.

[61] D. J. Lipman and W. R. Pearson. Rapid and sensitive protein similarity searches. *Science*, 227:1435–1441, 1985.

[62] D. J. Lockhart, H. Dong, M. C. Byrne, M. T. Follettie, M. V. Gallo, M. S. Chee, M. Mittmann, C. Wang, M. Kobayashi, H. Horton, and E. L. Brown. Expression monitoring by hybridization to high-density oligonucleotide arrays. *Nature Biotechnology*, 14:1675–1680, 1996.

[63] Y. Lysov, V. Florent'ev, A. Khorlin, K. Khrapko, V. Shik, and A. Mirzabekov. DNA sequencing by hybridization with oligonucleotides. *Doklady Academy Nauk USSR*, 303:1508–1511, 1988.

[64] D. Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM*, 25(2):322–336, 1978.

[65] B. Manthey. Non-approximability of weighted multiple sequence alignment. *Theoretical Computer Science*, 296(1):179–192, 2003.

[66] N. R. Markham and M. Zuker. DINAMelt web server for nucleic acid melting prediction. *Nucleic Acids Research*, 33:W577–81, 2005.

[67] W. J. Masek and M. S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980.

[68] O. V. Matveeva, S. A. Shabalina, V. A. Nemtsov, A. D. Tsodikov, R. F. Gesteland, and J. F. Atkins. Thermodynamic calculation and statistical correlations for oligo-probes design. *Nucleic Acids Research*, 31(14):4211–4217, 2003.

[69] D. P. Mehta and S. Sahni. *Data structures and applications*. CRC Press, 2005.

[70] D.W. Mount. *Bioinformatics: Sequence and Genome Analysis*. Cold Spring Harbor Laboratory Press, 2004.

[71] E. W. Myers. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *Bell System Technical Journal*, 29(2):147–160, 1950.

[72] G. Myers and R. Durbin. A table-driven, full-sensitivity similarity search algorithm. *Journal of Computational Biology*, 10:103–117, 2003.

[73] F. Naef and M. O. Magnasco. Solving the riddle of the bright mismatches: Labeling and effective binding in oligonucleotide arrays. *Physical Review E*, 68:011906, 2003.

[74] H. B. Jr Nicholas, A. J. Ropelewski, and D. W. Deerfield II. Strategies for multiple sequence alignment. *BioTechniques*, 32(3):317–330, 2002.

[75] P. Perrot. *A to Z of Thermodynamics*. Oxford University Press, 1998.

[76] A. Phillips, D. Janies, and W. Wheeler. Multiple sequence alignment in phylogenetic analysis. *Molecular Phylogenetics and Evoluton*, 16(3):572–591, 2000.

[77] Z. S. Qin. Clustering microarray gene expression data using weighted chinese restaurant process. *Bioinformatics*, 22(16):1988–1997, 2006.

[78] S. Rahmann. Rapid large-scale oligonucleotide selection for microarrays. In *Proceedings of the First Computational Systems Bioinformatics (CSB)*, pages 54–63, 2002.

[79] S. Rahmann. The shortest common supersequence problem in a microarray production setting. *Bioinformatics*, 19(suppl. 2):156–161, 2003.

[80] S. Rash and D. Gusfield. String Barcoding: Uncovering Optimal Virus Signatures. In *Proceedings of the Sixth Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, pages 254–261, 2002.

[81] K. Reinert, H. P. Lenhof, P. Mutzel, K. Mehlhorn, and J. D. Kececioglu. A branch-and-cut algorithm for multiple sequence alignment. In *Proceedings of the First Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, pages 241–250, 1997.

[82] A. Relogio, C. Schwager, A. Richter, W. Ansorge, and J. Valcarcel. Optimization of oligonucleotide-based DNA microarrays. *Nucleic Acids Research*, 30(11):e51, 2002.

[83] J-M. Rouillard, M. Zuker, and E. Gulari. OligoArray2.0: design of oligonucleotide probes for DNA microarrays using a thermodynamic approach. *Nucleic Acids Research*, 31(12):3057–3062, 2003.

[84] W. Rychlik, W. J. Spencer, and R. E. Rhoads. Optimization of the annealing temperature for DNA amplification in vitro. *Nucleic Acids Research*, 18(21):6409–6412, 1990.

[85] J. J. SantaLucia, H. T. Allawi, and P. A. Seneviratne. Improved nearest-neighbor parameters for predicting DNA duplex stability. *Biochemistry*, 35(11):3555–3562, 1996.

[86] M. Schena. *Microarray Analysis*. Hoboken, NJ: Wiley-Liss. 630p, 2003.

[87] R. Sengupta and M. Tompa. Quality control in manufacturing oligo arrays: A combinatorial design approach. *Journal of Computational Biology*, 9(1):1–22, 2002.

[88] D. K. Slonim, P. Tamayo, J. P. Mesirov, T. R. Golub, and E. S. Lander. Class prediction and discovery using gene expression data. In *Proceedings of the Fourth Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, pages 263–272, 2000.

[89] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[90] W. K. Sung and W. H. Lee. Fast and accurate probe selection algorithm for large genomes. In *Proceedings of the Second Computational Systems Bioinformatics (CSB)*, pages 65–74, 2003.

[91] T.G. Szymanski. *A special case of the maximal common subsequence problem.* Technical Report TR-170, Computer Science Laboratory, Princeton University, 1975.

[92] A. C. Tolonen, D. F. Albeanu, J. F. Corbett, and H. Handley. Optimized *in situ* construction of oligomers on an array surface. *Nucleic Acids Research*, 30(20):e107, 2002.

[93] A. C. Tolonen, D. F. Albeanu, J. F. Corbett, H. Handley, C. Henson, and P. Malik. Optimized *in situ* construction of oligomers on an array surface. *Nucleic Acids Research*, 30(20):e107, 2003.

[94] J. D. Watson and F. H. C. Crick. Molecular structure of nucleic acids: A structure for Deoxyribose Nucleic Acid. *British journal Nature*, 171:737–738, 1953.

[95] J. G. Wetmur. DNA probes: applications of the principles of nucleic acid hybridization. *Critical Reviews in Biochemistry and Molecular Biology*, 26(3-4):227–59, 1991.

[96] M. A. Wright and G. M. Church. An open-source oligomicroarray standard for human and mouse. *Nature Biotechnology*, 20:1082–1083, 2002.

[97] B. Y. Wu, G. Lancia, V. Bafna, K. M. Chao, R. Ravi, and C. Y. Tang. A polynomial-time approximation scheme for minimum routing cost spanning trees. *SIAM Journal on Computing*, 29(3):761–778, 1999.

[98] Z. Wu and R. A. Irizarry. Stochastic models inspired by hybridization theory for short oligonucleotide microarrays. *Journal of Computational Biology*, 12:882–893, 2005.

[99] Y. Xu, V.Olman, and D. Xu. Minimum spanning trees for gene expression data clustering. *Genome Informatics*, 12:24–33, 2001.

[100] Y. Xu, V.Olman, and D. Xu. Clustering gene expression data using a graph-theoretic approach: an application of minimum spanning trees. *Bioinformatics*, 18(4):536–545, 2002.

[101] K. Y. Yeung, M. Medvedovic, and R. E. Bumgarner. Clustering gene-expression data with repeated measurements. *Genome Biology*, 4:R34, 2003.

[102] M. Zuker, D. H. Mathews, and D. H. Turner. *Algorithms and Thermodynamics for RNA Secondary Structure Prediction: A Practical Guide.* NATO ASI Series, Kluwer Academic publishers, Dordrecht, NL, 1999.

Index